INFORMATION RETRIEVAL

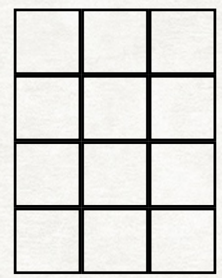
Luca Manzoni
lmanzoni@units.it

Lecture 10

LECTURE OUTLINE

TODAY WITH MATRICES

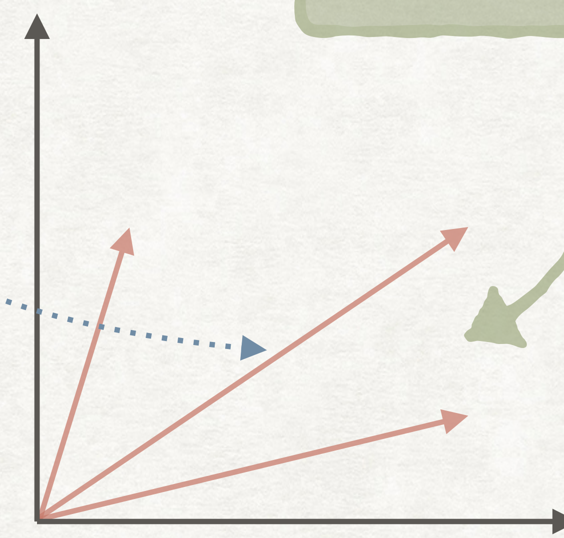
Matrix Decomposition



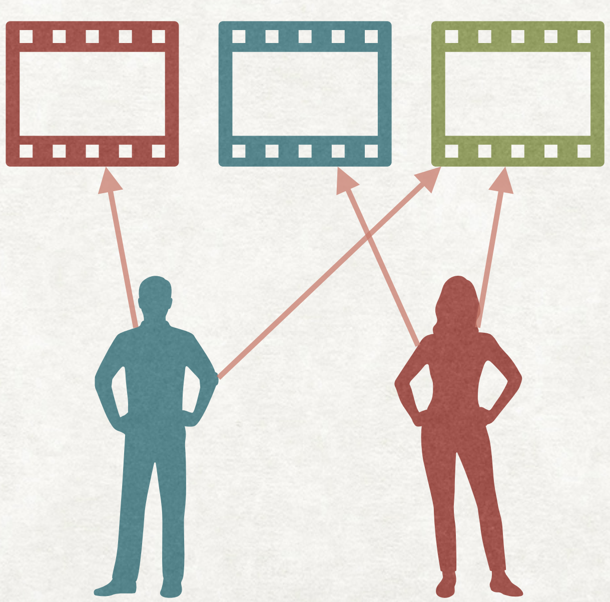
$= U \, \Sigma \, V^T$

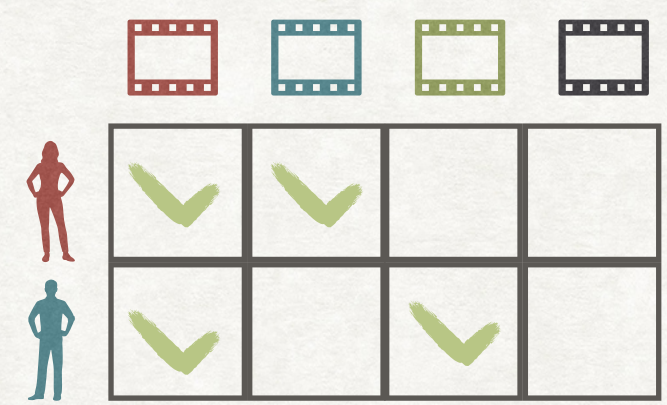
Latent Semantic Indexing

$(x_1, x_2, \ldots, x_n)$



Recommender Systems





Matrix Factorisation

MATRIX DECOMPOSITION

A BRIEF RECAP

ASSUMING KNOWLEDGE OF EIGENVALUES

- We want to write a matrix as a product of other matrices…
- …usually with some “interesting” properties.
- We will recall two matrix decompositions:
  - Symmetric diagonal decomposition
  - Singular value decomposition (SVD)
- We recall how SVD can be used to provide an approximation of the original matrix.

SYMMETRIC DIAGONAL DECOMPOSITION

Let S be a square $M \times M$ matrix which is:

- Real-valued
- Symmetric
- With M linearly independent eigenvectors

Then there exists a **symmetric diagonal decomposition**:

$$S = Q \, \Lambda \, Q^T$$

SYMMETRIC DIAGONAL DECOMPOSITION

$$S = Q \, \Lambda \, Q^T$$

Where:

- The columns of Q are orthogonal eigenvectors of S
- All columns of Q are of vectors of unit length
- All entries of Q are real-valued
- Λ is the *diagonal* matrix containing the eigenvalues of Q in the diagonal (by convention in non-increasing order)

THE TERM-DOCUMENT MATRIX

Documents in the collection

Terms in the collection

$$
\begin{array}{c c}
 & \begin{array}{ccc} d_1 & d_2 & d_3 \end{array} \\
\begin{array}{c} t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} & \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}
\end{array}
$$

The term t_4 is present in the document d_3

Actually, the value in row i and column j can be any “weighting”.
For example the $\mathrm{tf\text{-}idf}$ for term t_i in the document d_j.

THE TERM-DOCUMENT MATRIX

Some issues with the term-document matrix:

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

- Not square
- Not symmetric

We will need another method to perform a matrix decomposition of C, since the symmetrical diagonal decomposition is not applicable

SINGULAR VALUE DECOMPOSITION

Given a real-valued matrix C with M rows and N columns of rank $r \leq \min\{M, N\}$, and let:

- U be the $M \times r$ matrix with the orthonormal eigenvectors of CC^T as columns.
- V be the $r \times N$ matrix with the orthonormal eigenvectors of C^TC as columns.

Then C can be written as:

$$C = U\Sigma V^T$$

SINGULAR VALUE DECOMPOSITION

$$C = U\Sigma V^T$$

where:

- The eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_r$ are the same for CC^T and C^TC.
- $\lambda_1, \lambda_2, \ldots, \lambda_r$ are in *non-increasing* order.
- The matrix Σ is a square $r \times r$ matrix containing in the diagonal all $\sqrt{\lambda_i}$, called the singular values of C.

SVD FOR THE TERM-DOCUMENT MATRIX

$$U = \begin{bmatrix} -0.436 & 0.707 & 0.408 \\ -0.436 & 0 & -0.816 \\ -0.655 & 0 & 0 \\ -0.436 & -0.707 & 0.408 \end{bmatrix}$$

Left singular vectors

$$\Sigma = \begin{bmatrix} 2.646 & 0 & 0 \\ 0 & 0.999 & 0 \\ 0 & 0 & 0.999 \end{bmatrix}$$

The values

$$[2.646 \quad 0.999 \quad 0.999]$$

Are called the singular values of C

$$V^T = \begin{bmatrix} -0.577 & -0.577 & -0.577 \\ 0 & 0.707 & -0.707 \\ 0.816 & -0.408 & -0.408 \end{bmatrix}$$

Right singular vectors

THE TERM-DOCUMENT MATRIX

We can consider the matrix CC^T:

$$
\begin{array}{c c}
 & \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\
\begin{array}{c} t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} & \begin{bmatrix} 2 & 1 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 2 & 2 & 3 & 2 \\ 1 & 1 & 2 & 2 \end{bmatrix}
\end{array}
$$

Number of documents where t_4 and t_2 co-occur

Actually, the value in row i and column j is, depending on how C is constructed, some "measure" of co-occurrence of the terms t_i and t_j

SOME "STUFF" TO NOTICE

LINKING SVD WITH SYMMETRIC DIAGONAL DECOMPOSITION

Take CC^T

Rewrite C as $U\Sigma V^T$

Rewrite C^T as $V\Sigma^T U^T$

You get $U\Sigma V^T V\Sigma^T U^T$

Which is $U\Sigma^2 U^T$

In some sense we can view looking at co-occurrence of terms can be interpreted as "working" in the space of terms (which we reach using U)

THE TERM-DOCUMENT MATRIX

We can also consider the matrix $C^T C$:

$$\begin{array}{cc} & \begin{array}{ccc} d_1 & d_2 & d_3 \end{array} \\ \begin{array}{c} d_1 \\ d_2 \\ d_3 \end{array} & \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & 2 \\ 2 & 2 & 3 \end{bmatrix} \end{array}$$

Number of terms in common between document d_3 and d_2

Actually, the value in row i and column j is, depending on how C is constructed, some "measure" of "overlap" between d_i and d_j

LOW-RANK APPROXIMATION

BASICS

- The main idea is that we can reduce the “space occupied” by a matrix by reducing its rank…
- …however we want to minimise the error introduced by the approximation.
- SVD provides a way to efficiently perform this approximation.
- At least with respect to the **Frobenius norm**:

$$\|X\|_F = \sum_{i=1}^{M} \sum_{j=1}^{N} X_{i,j}^2$$

LOW RANK APPROXIMATIONS WITH SVD

ZEROING OUT SINGULAR VALUES

Given a real-valued matrix C, compute its SVD decomposition $U\Sigma V^T$

Let $\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_r}$ be the r singular values of C

Fix $k \in \mathbb{N}$ as the rank of the approximation C_k that we want to compute.

Build Σ_k starting from Σ by zeroing out the smallest $r - k$ singular values (i.e., only $\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_k}$ remains).

Let the approximation C_k be $U\Sigma_k V^T$.

LOW RANK APPROXIMATIONS WITH SVD

ZEROING OUT SINGULAR VALUES

$$\Sigma = \begin{bmatrix} 2.646 & 0 & 0 \\ 0 & 0.999 & 0 \\ 0 & 0 & 0.999 \end{bmatrix} \xleftarrow{\text{Compute SVD}} C = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Keep only two singular values

$$\Sigma_2 = \begin{bmatrix} 2.646 & 0 & 0 \\ 0 & 0.999 & 0 \\ 0 & 0 & 0 \end{bmatrix} \xrightarrow{U\Sigma_2 V^T} C_2 = \begin{bmatrix} 0.667 & 1.667 & 0.667 \\ 0.667 & 0.667 & 0.667 \\ 1 & 1 & 1 \\ 0.667 & 0.167 & 1.167 \end{bmatrix}$$

Is this a good approximation?

Across all matrices of rank two, C_2 minimises $\|C - C_2\|_F$

LOW RANK APPROXIMATION

WHAT WE NEED TO MEMORISE

$$U = \begin{bmatrix} -0.436 & 0.707 & 0.408 \\ -0.436 & 0 & -0.816 \\ -0.655 & 0 & 0 \\ -0.436 & -0.707 & 0.408 \end{bmatrix} \qquad \Sigma_2 = \begin{bmatrix} 2.646 & 0 & 0 \\ 0 & 0.999 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad V^T = \begin{bmatrix} -0.577 & -0.577 & -0.577 \\ 0 & 0.707 & -0.707 \\ 0.816 & -0.408 & -0.408 \end{bmatrix}$$

No need to memorise this column

No need to memorise this row

We can rewrite everything as a "truncated" SVD $U'_k \Sigma'_k V'^T_k$:

$$\begin{bmatrix} -0.436 & 0.707 \\ -0.436 & 0 \\ -0.655 & 0 \\ -0.436 & -0.707 \end{bmatrix} \begin{bmatrix} 2.646 & 0 \\ 0 & 0.999 \end{bmatrix} \begin{bmatrix} -0.577 & -0.577 & -0.577 \\ 0 & 0.707 & -0.707 \end{bmatrix}$$

LATENT SEMANTIC INDEXING

LATENT SEMANTIC INDEXING

MAIN IDEAS

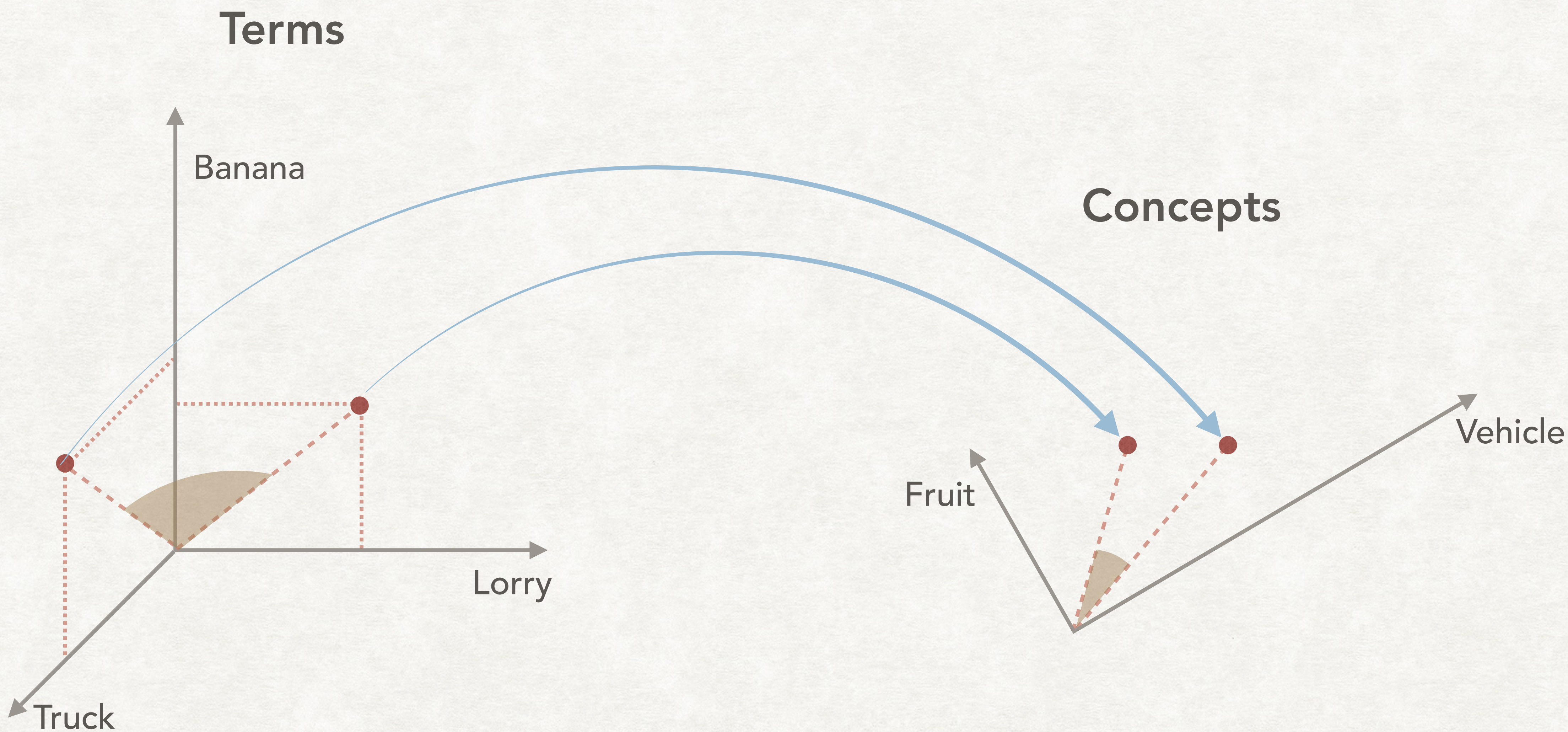
- Recall that the vector space representation does not address two issues:
  - **Synonymy**. E.g., when searching for “laptop” we do not find the documents that use “notebook”
  - **Polysemy**. When the same word is used with multiple meanings.
- We can potentially use a large thesaurus for the first problem…
- …or we can use the co-occurrence of terms to try to solve the problems automatically.

HOW TO USE THE SVD

TERMS, DOCUMENTS, AND CONCEPTS

- We use the SVD as a way to represent documents in a reduced space.
- Instead of using terms as the basis of out vector space, we will employ “pseudo-terms”.
- Dimensionality reduction is used to provide a compact representation of the the documents and queries.
- The main idea is that we map terms to concepts (i.e., how much each term represents a certain concept)…
- …and then concepts to documents (i.e., how much each document contains a certain concept).

MAIN IDEA
(GRAPHICALLY)



Two documents use different terms for the same concept.
If we remap everything in a space where the axes represent concepts the two documents will have a higher similarity.

LET'S GO BACK TO THE SVD

$$U = \begin{bmatrix} -0.436 & 0.707 & 0.408 \\ -0.436 & 0 & -0.816 \\ -0.655 & 0 & 0 \\ -0.436 & -0.707 & 0.408 \end{bmatrix}$$

U is the term-concept matrix

Each column represents how much each term is represented by a certain concept

$$\Sigma = \begin{bmatrix} 2.646 & 0 & 0 \\ 0 & 0.999 & 0 \\ 0 & 0 & 0.999 \end{bmatrix}$$

Σ is the concept matrix

Each value represents the "weight" of a concept

$$V^T = \begin{bmatrix} -0.577 & -0.577 & -0.577 \\ 0 & 0.707 & -0.707 \\ 0.816 & -0.408 & -0.408 \end{bmatrix}$$

V is the document-concept matrix

Each row (column in V^T) represents how much a document contains a certain concept.

LET'S GO BACK TO THE SVD

$$U = \begin{bmatrix} -0.436 & 0.707 & 0.408 \\ -0.436 & 0 & -0.816 \\ -0.655 & 0 & 0 \\ -0.436 & -0.707 & 0.408 \end{bmatrix}$$

The left singular vectors
are pseudo-terms

$$\Sigma = \begin{bmatrix} 2.646 & 0 & 0 \\ 0 & 0.999 & 0 \\ 0 & 0 & 0.999 \end{bmatrix}$$

$$V^T = \begin{bmatrix} -0.577 & -0.577 & -0.577 \\ 0 & 0.707 & -0.707 \\ 0.816 & -0.408 & -0.408 \end{bmatrix}$$

The **columns** of V^T are
a representation
of the documents
using the pseudo-terms

PSEUDOTERMS

AN EXAMPLE

$$
\begin{array}{r c}
 & \begin{array}{ccc} d_1 & d_2 & d_3 \end{array} \\
\text{CAT} \rightarrow & \\
\text{DOG} \rightarrow & \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \\
\text{TRUCK} \rightarrow & \\
\text{BANANA} \rightarrow &
\end{array}
$$

Second pseudo-term

$$
U = \begin{bmatrix} -0.436 & \boxed{0.707} & 0.408 \\ -0.436 & 0 & -0.816 \\ -0.655 & 0 & 0 \\ -0.436 & -0.707 & 0.408 \end{bmatrix}
$$

It represents the concept
$0.707 \times$ CAT – $0.707 \times$ BANANA

While we might hope to obtain things like $0.75 \times$ truck $+ 0.25 \times$ car to represent concepts like "vehicle", the construction of the pseudo-terms totally depends on the term-document matrix, i.e., on the collection.

LATENT SEMANTIC INDEXING

REMAPPING DOCUMENTS

- A remapped document $\hat{d}_i$ is a column of the matrix V^T.
- To obtain the original document we perform $d_i = U\Sigma\,\hat{d}_i$.
- Which means that if we want to remap a document in its reduce form we have to compute:
  - $(U\Sigma)^{-1}d_i = (U\Sigma)^{-1}U\Sigma\,\hat{d}_i$ (multiply by the inverse of $U\Sigma$)
  - $\Sigma^{-1}U^{-1}d_i = \hat{d}_i$ (recall that $(AB)^{-1} = B^{-1}A^{-1}$)
  - $\hat{d}_i = \Sigma^{-1}U^T d_i$ (since the inverse of U is U^T)

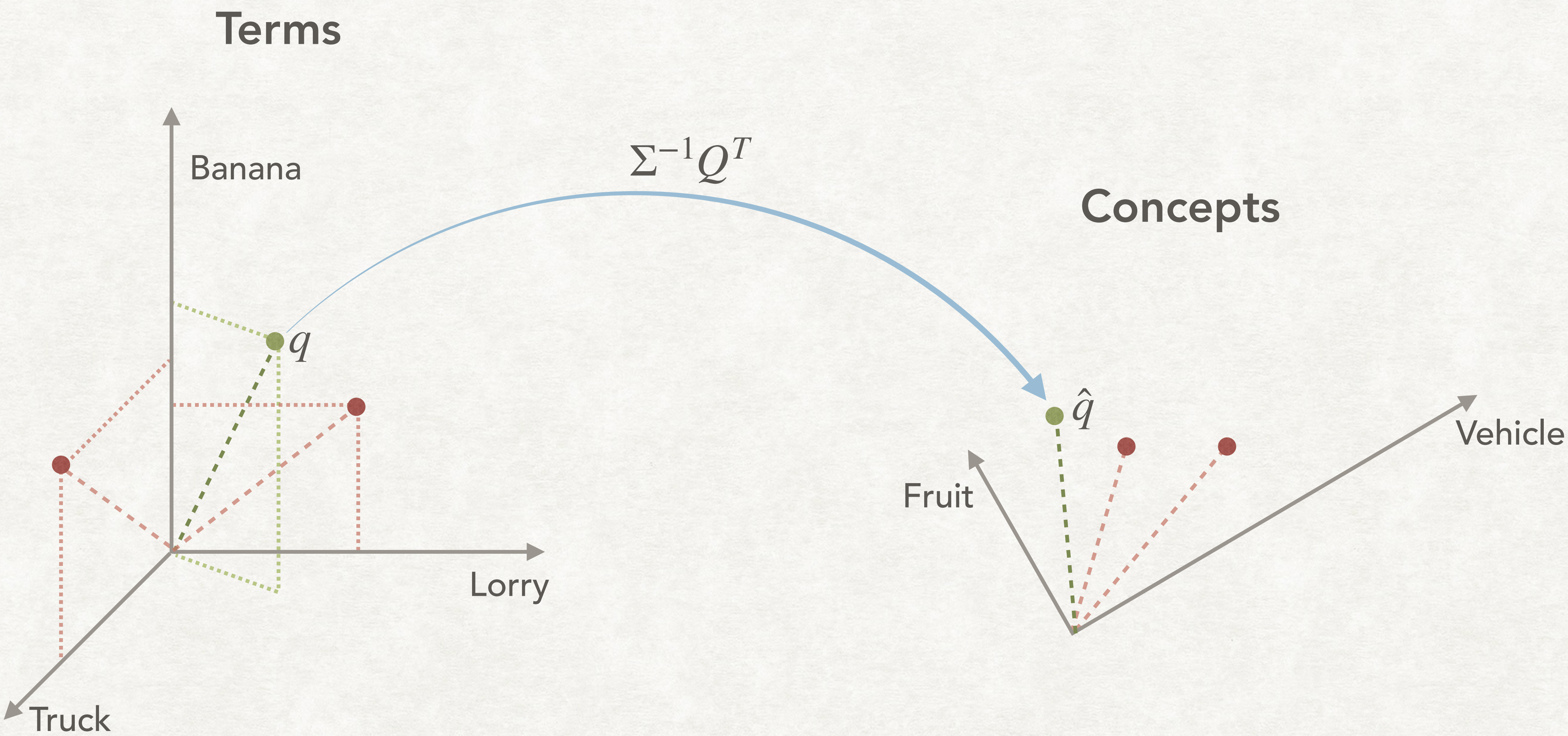
LATENT SEMANTIC INDEXING

REMAPPING DOCUMENTS

- We can now remap documents by multiplying them by $\Sigma^{-1}U^T$.
- We can reduce the dimensionality of the "concepts space" by selecting $k \in \mathbb{N}$ and using Σ_k' and U_k'
- k represents the number of "important concepts" to keep. Usually a few hundreds.
- How about queries? Like in the vector space model they are like documents.
- Given a query q, the remapped query is $\hat{q} = \Sigma^{-1}U^T q$.

QUERIES

(GRAPHICALLY)



We remap the query and compute the similarity in the reduced space (for example with cosine similarity)

ADDING DOCUMENTS

NOT AS EASY

- To add a document d in the standard vector space model is easy.
- To store it in this remapped/reduced representation we must remap it first: $\hat{d} = \Sigma^{-1} Q^T d$.
- However, the space of concept has been generated starting from the initial collection.
- While we add documents the concepts can change, thus we might see a degradation of the quality of the retrieval as more documents are added.
- In that case we might need to create a new mapping.

THE GOOD, THE BAD, AND THE UGLY

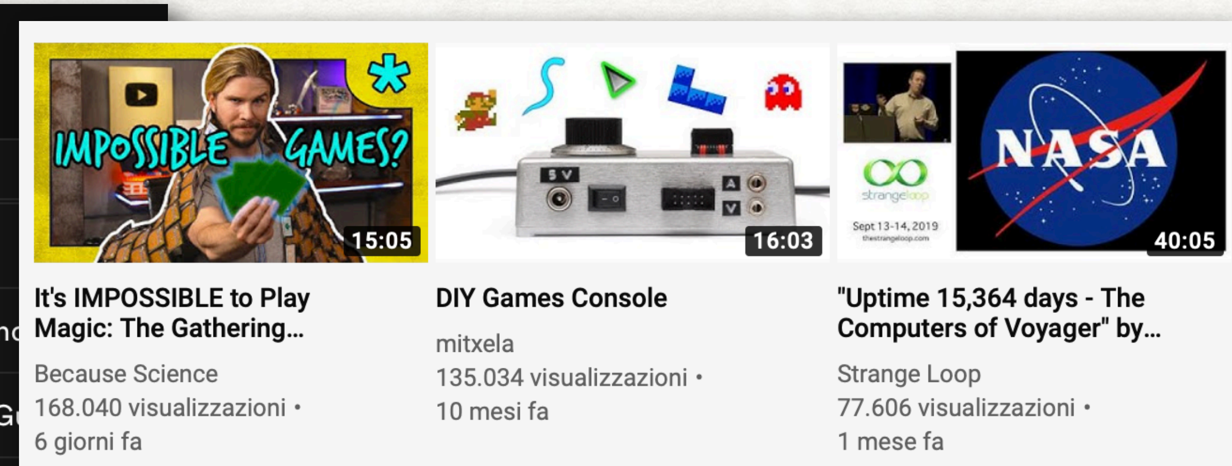
- Using the latent semantic indexing we can address the problems of synonymity and polysemy.
- By using "concepts" instead of terms we can improve the quality of the retrieval.
- However, computing the SVD is expensive and re-computing it when sufficiently new documents arrive is necessary.
- We can use the same mapping for other tasks: finding synonyms, clustering documents according to topics (e.g., with k-means), expand a query by adding similar terms, etc.

RECOMMENDER SYSTEMS

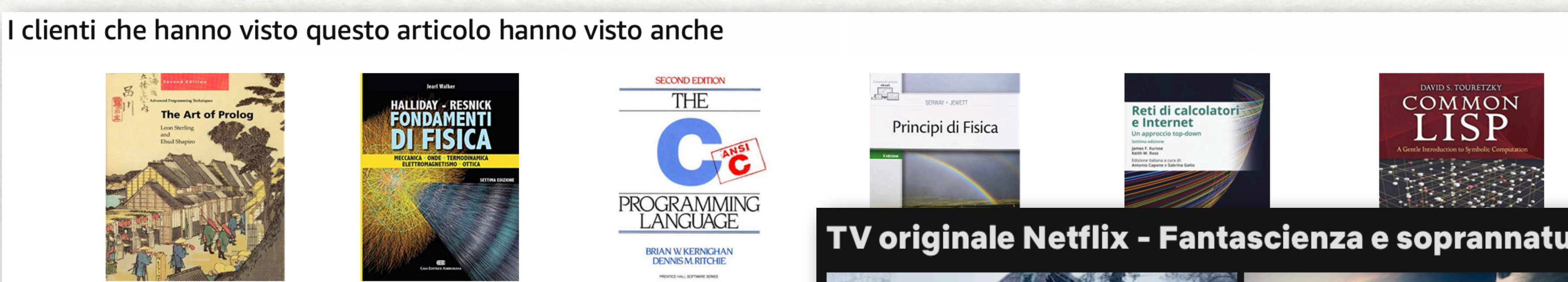
EXAMPLE OF USES OF RECOMMENDER SYSTEMS

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.

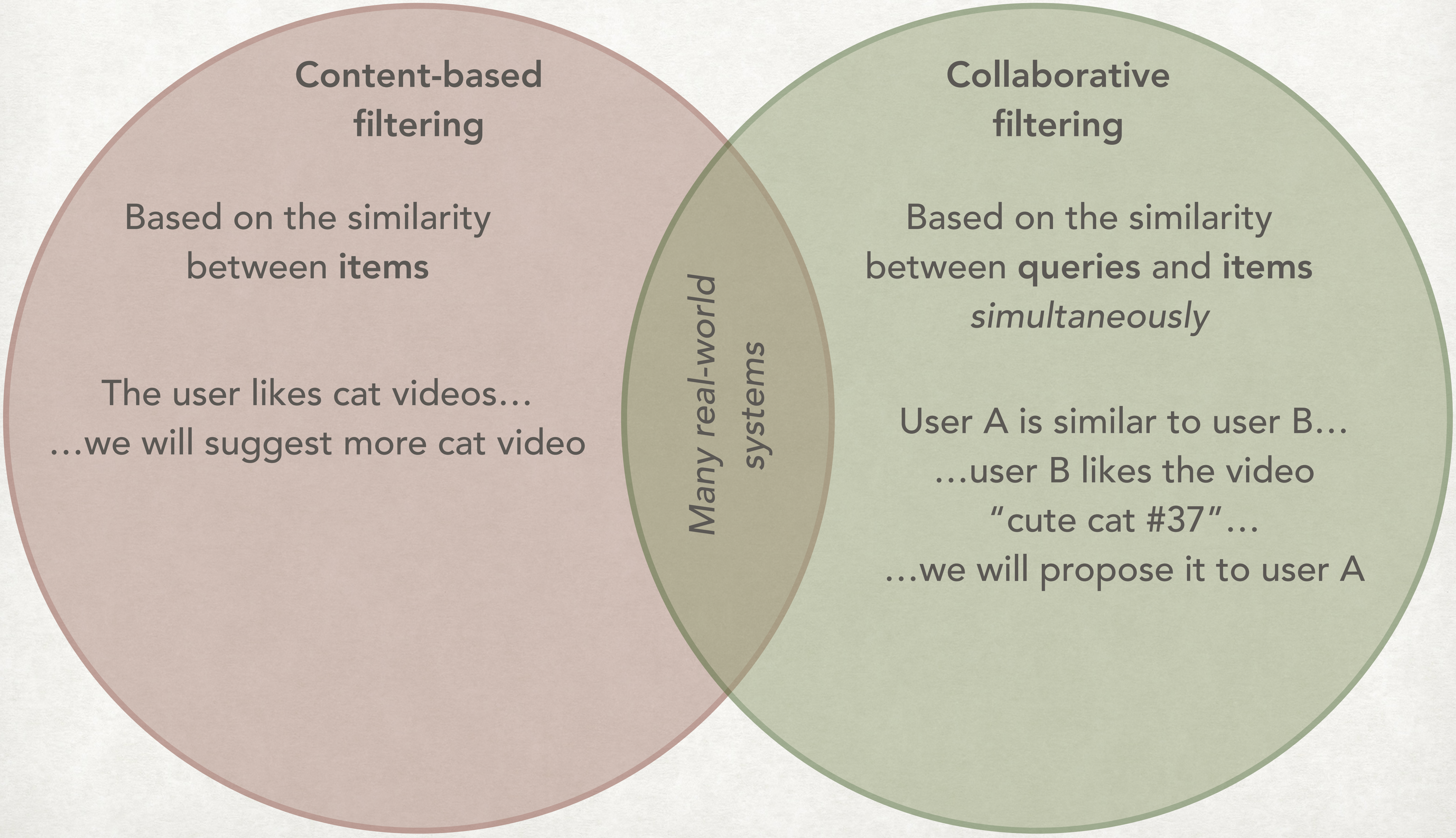
BASIC CHARACTERISTICS

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

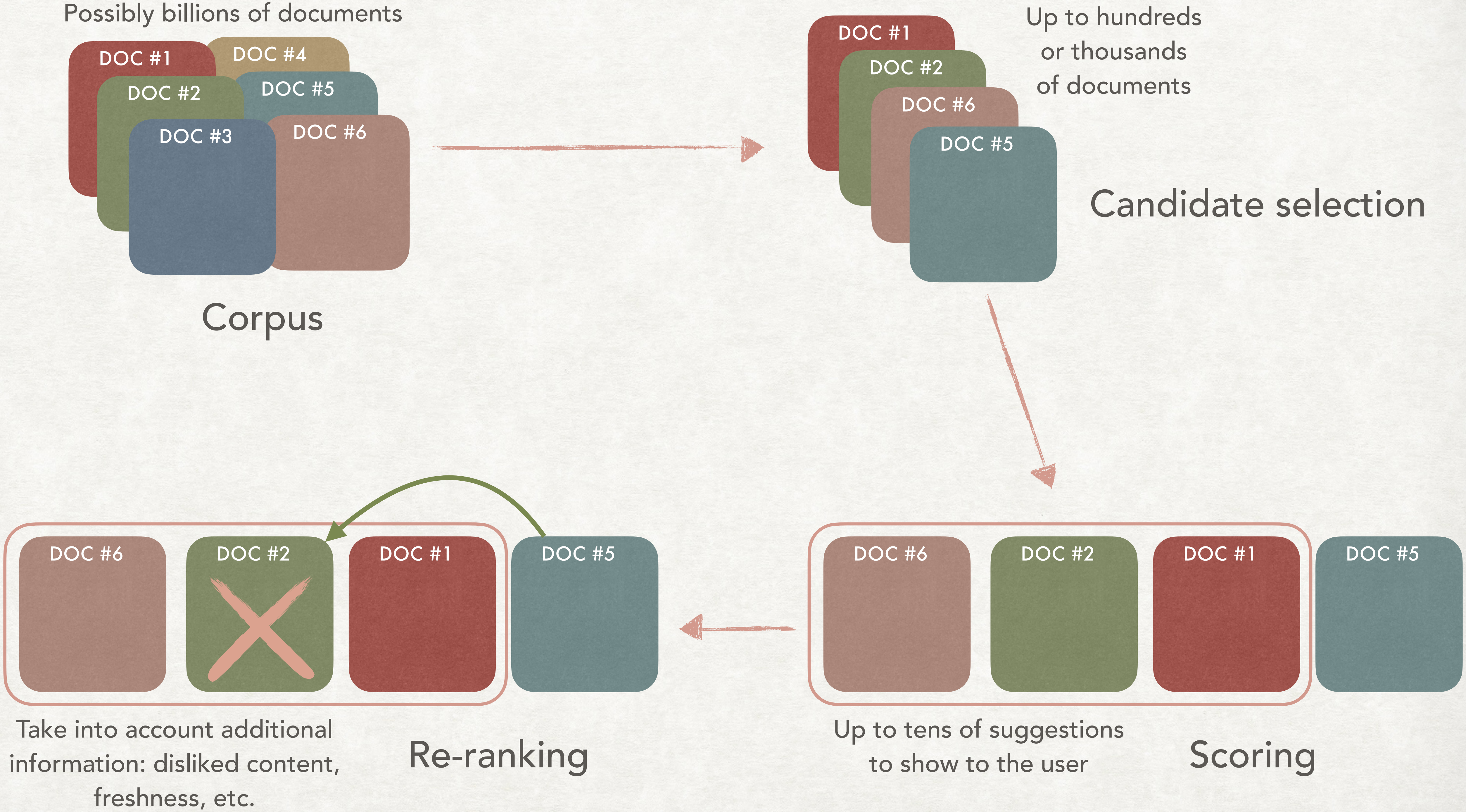


PROBLEMS FOR RECOMMENDER SYSTEMS

- 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.

STRUCTURE OF A RECOMMENDER SYSTEM

AN EXAMPLE FROM GOOGLE



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.

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.
- First of all, 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 $\displaystyle\sum_{i,j \in \text{Obs}} (C_{i,j} - C'_{i,j})^2$

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

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

The algorithm is guaranteed to converge and can be parallelised.