Singular value decomposition

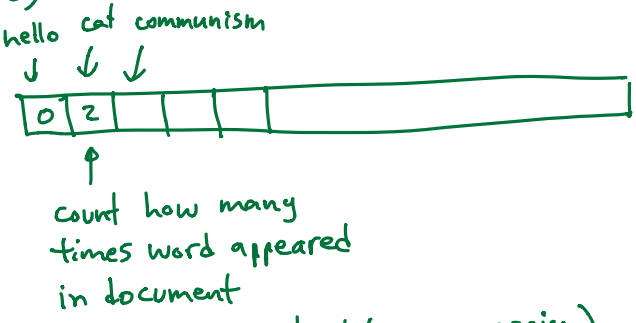
DSE 220

Word embeddings

1. Traditional representation of documents: bag of words

V: vocab (eg. 50,000 English words)

document $\longrightarrow$ vector in $\mathbb{R}^{50000}$



The good:

① This allows us to use standard classification methods (eg. logistic regression) for vector data.

② Has been reasonably successful in many applications.

The bad:

① Very high dimensional (i.e. need lots of training data).

Possible saving grace: many of these words are very close in meaning:

cat — kitten — kitty — pussycat — ...

and many are fairly close, eg:

- different dinner foods

→ We should be able to generalize across these words

This is what word embeddings exploit.

2. Embeddings

$\varphi : V \longrightarrow \mathbb{R}^{300}$, ie. map each word w to a 300-dim vector $\varphi(w)$

Goal: Words w, w' are "similar" $\iff$ $\varphi(w)$ is close to $\varphi(w')$

(more precisely, if $\varphi(w) \cdot \varphi(w')$ is large)

E.g. $\varphi(\text{cat}) \approx \varphi(\text{kitten})$

ⓐ Words are similar if they appear in similar contexts

E.g. I had steak and potatoes for dinner

I had sausage and potatoes for dinner

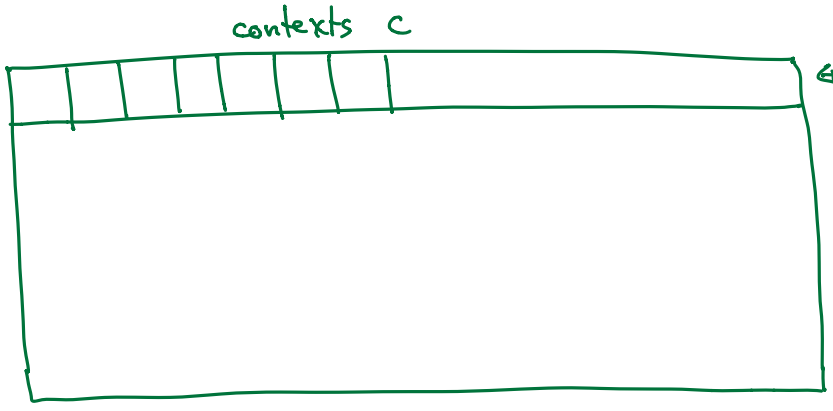
(b) Look at the contexts in which words appear

sentence: — — w — —

two words before

two words after

Matrix M



M is huge:

- 50,000 words (say)
- Maybe 10^6 contexts?

} 50000×10^6

Each row M_w is a vector representation of the corresponding word that captures its meaning.

(c) Problem: M_w is huge: 10^6-dimensional.

Use PCA to reduce dimension to (say) 300.

This gives us an embedding of V.

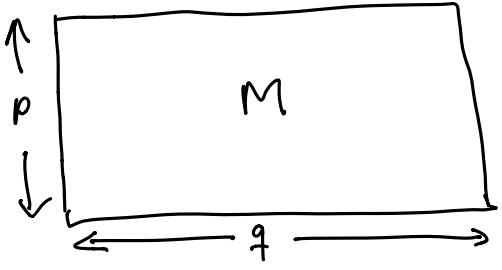
Generalizing the spectral decomposition

SVD: a popular way of constructing embeddings.

For **symmetric** matrices (e.g. covariance matrices), we have seen:

- Results about existence of eigenvalues and eigenvectors
- Eigenvectors form an alternative basis
- Resulting spectral decomposition, used in PCA

What about **arbitrary** matrices $M \in \mathbb{R}^{p \times q}$?



Data that we want to "embed"

Singular value decomposition (SVD)

M: p × q

Any $p \times q$ matrix ($p \leq q$) has a **singular value decomposition**:

$$M = \underbrace{\begin{pmatrix} \uparrow & & \uparrow \\ u_1 & \cdots & u_p \\ \downarrow & & \downarrow \end{pmatrix}}_{p \times p \text{ matrix } U} \underbrace{\begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_p \end{pmatrix}}_{p \times p \text{ matrix } \Lambda} \underbrace{\begin{pmatrix} \longleftarrow & v_1 & \longrightarrow \\ & \vdots & \\ \longleftarrow & v_p & \longrightarrow \end{pmatrix}}_{p \times q \text{ matrix } V^T}$$

p × q

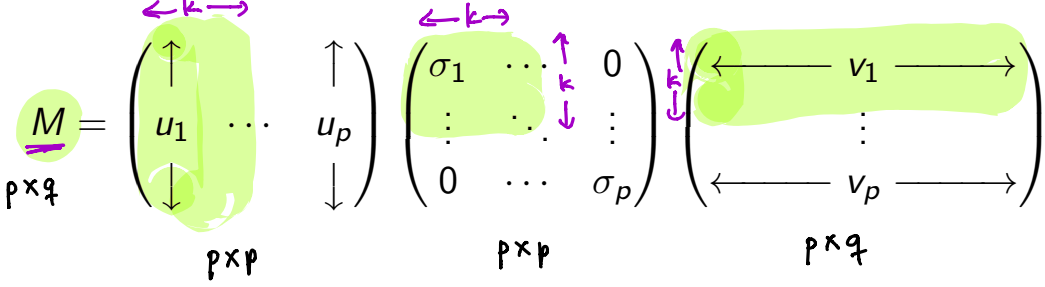
- $u_1, \ldots, u_p$ are orthonormal vectors in $\mathbb{R}^p$
- $v_1, \ldots, v_p$ are orthonormal vectors in $\mathbb{R}^q$
- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ are **singular values**

(σ_1, u_1, v_1)
(σ_2, u_2, v_2)
(σ_3, u_3, v_3)

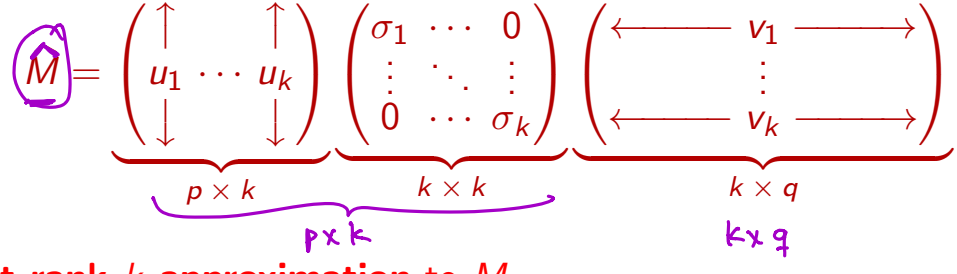
One line of Python ... ("thin SVD")

Low-rank approximation

Singular value decomposition of $p \times q$ matrix M (with $p \leq q$):



A concise approximation to M, for any $k \leq p$



$\widehat{M}$ is the **best rank-k approximation** to M.

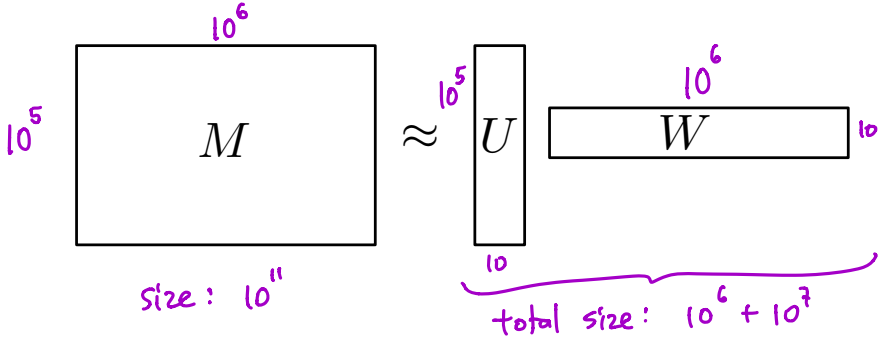
Optimality property

Let M be any $p \times q$ matrix.

Want to approximate M by a $p \times q$ matrix $\widehat{M}$ of the form UW:

- U is $p \times k$ and W is $k \times q$
- $k \leq p, q$ is of our choosing

$k = 10$

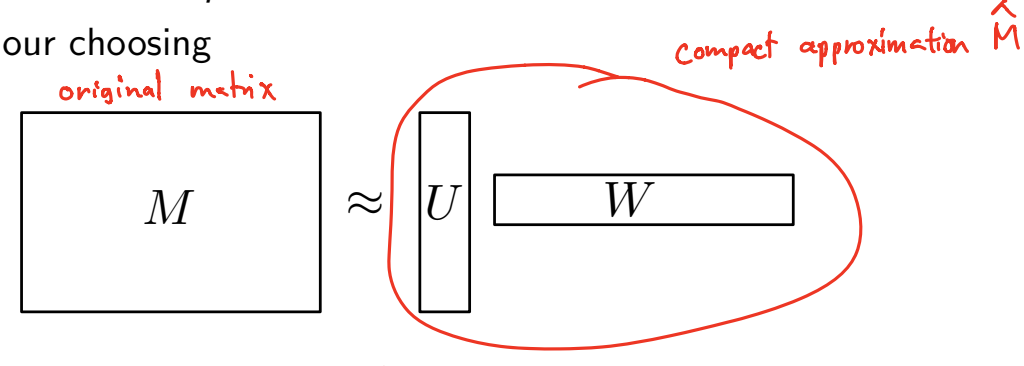


Optimality property

Let M be any $p \times q$ matrix.

Want to approximate M by a $p \times q$ matrix $\widehat{M}$ of the form UW:

- U is $p \times k$ and W is $k \times q$
- $k \leq p, q$ is of our choosing



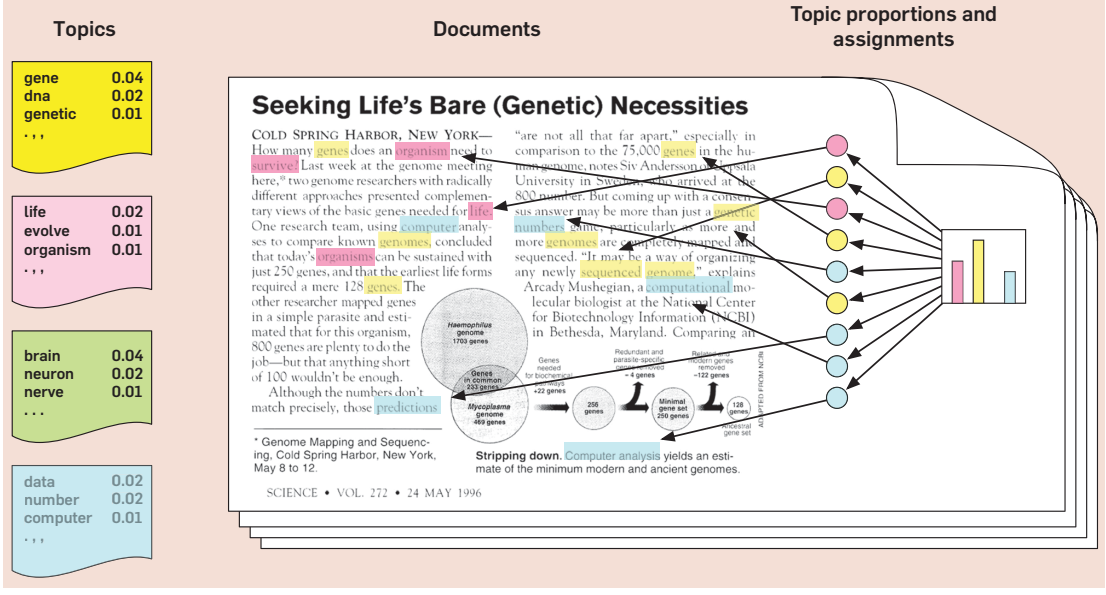
SVD yields the best such approximation $\widehat{M}$, minimizing the squared error

$$\sum_{i,j} (M_{ij} - \widehat{M}_{ij})^2.$$

Example: Topic modeling

Eg. Corpus of all Science articles over 100 years

Blei (2012):



genetics →

evolution →

neuroscience →

computer science →

K topics

Embedding of document: combination of topics → vector in $\mathbb{R}^k$

Latent semantic indexing (LSI)

Given a large corpus of n documents:

- Fix a vocabulary, say of V words.
- Bag-of-words representation for documents: each document becomes a vector of length V, with one coordinate per word.
- The corpus is an $n \times V$ matrix, one row per document.

n = #docs, V = #words

Huge matrix

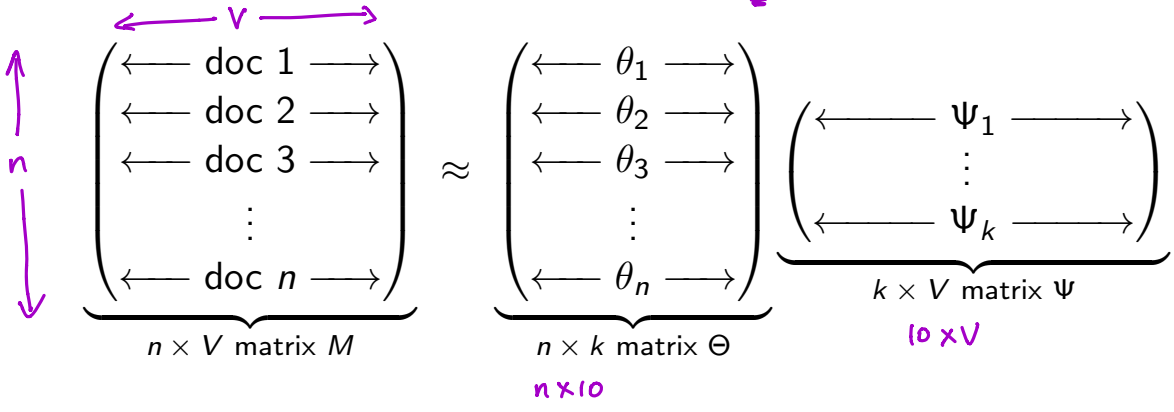
	cat	dog	house	boat	garden	...
Doc 1	4	1	1	0	2	
Doc 2	0	0	3	1	0	
Doc 3	0	1	3	0	0	
		⋮				

Let's find a concise approximation to this matrix M.

Latent semantic indexing, cont'd

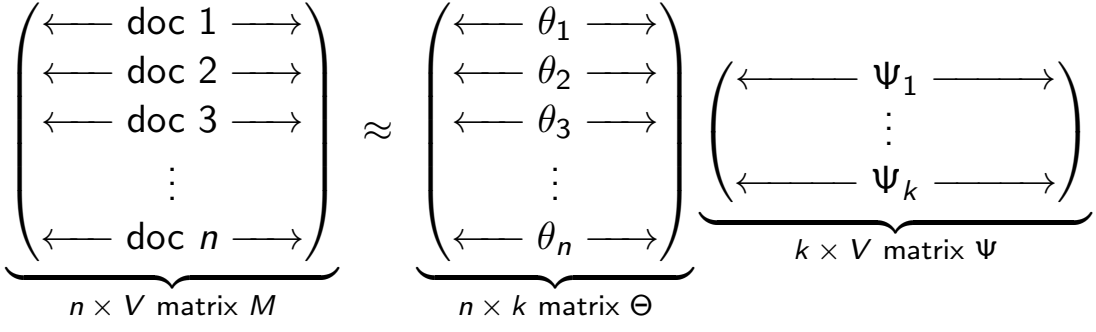
Use SVD to get an approximation to M: for small k,

e.g. $k = 10$



Latent semantic indexing, cont'd

Use SVD to get an approximation to M: for small k,

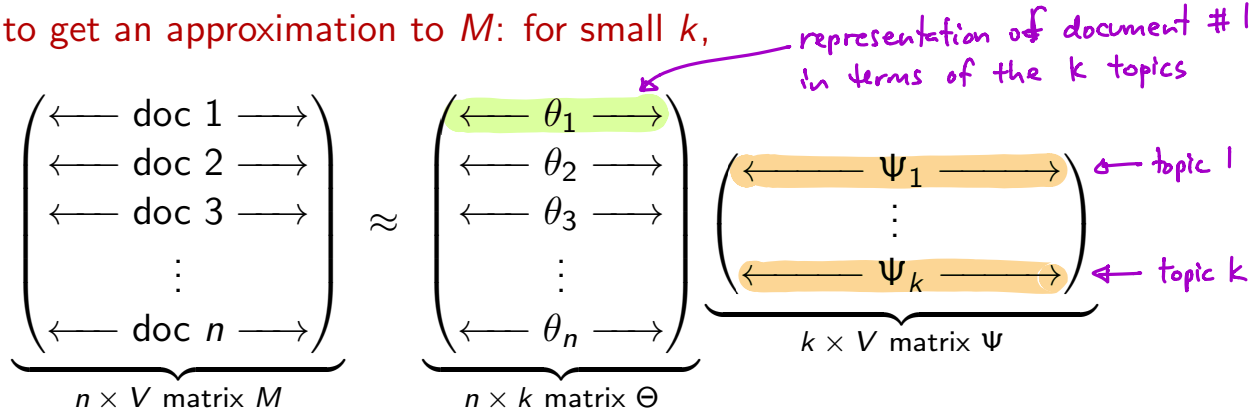


Think of this as a *topic model* with k topics.

- Ψ_j is a vector of length V describing topic j: coefficient Ψ_{jw} is large if word w appears often in that topic.
- Each document is a combination of topics: θ_{ij} = weight of topic j in doc i.

Latent semantic indexing, cont'd

Use SVD to get an approximation to M: for small k,



Think of this as a *topic model* with k topics.

- Ψ_j is a vector of length V describing topic j: coefficient Ψ_{jw} is large if word w appears often in that topic.
- Each document is a combination of topics: θ_{ij} = weight of topic j in doc i.

Document i originally represented by ith row of M, a vector in $\mathbb{R}^V$.
Can instead use $\theta_i \in \mathbb{R}^k$, a more concise "semantic" representation.

Example: Collaborative filtering

Details and images from Koren, Bell, Volinksy (2009).

Recommender systems: matching customers with products.

- Given: data on prior purchases/interests of users
- Recommend: further products of interest

Prototypical example: Netflix.

Example: Collaborative filtering

Details and images from Koren, Bell, Volinksy (2009).

Recommender systems: matching customers with products.

- Given: data on prior purchases/interests of users
- Recommend: further products of interest

Prototypical example: Netflix.

A successful approach: **collaborative filtering**.

- Model dependencies between different products, and between different users.
- Can give reasonable recommendations to a relatively new user.

Example: Collaborative filtering

Details and images from Koren, Bell, Volinksy (2009).

Recommender systems: matching customers with products.

- Given: data on prior purchases/interests of users
- Recommend: further products of interest

Prototypical example: Netflix.

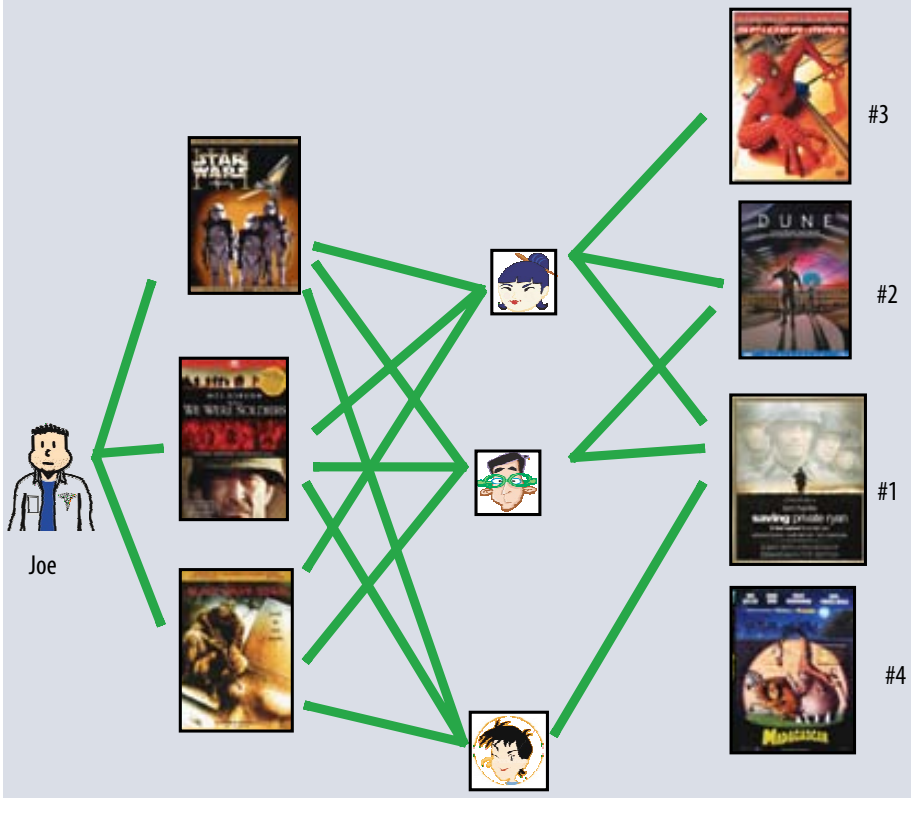
A successful approach: **collaborative filtering**.

- Model dependencies between different products, and between different users.
- Can give reasonable recommendations to a relatively new user.

Two strategies for collaborative filtering:

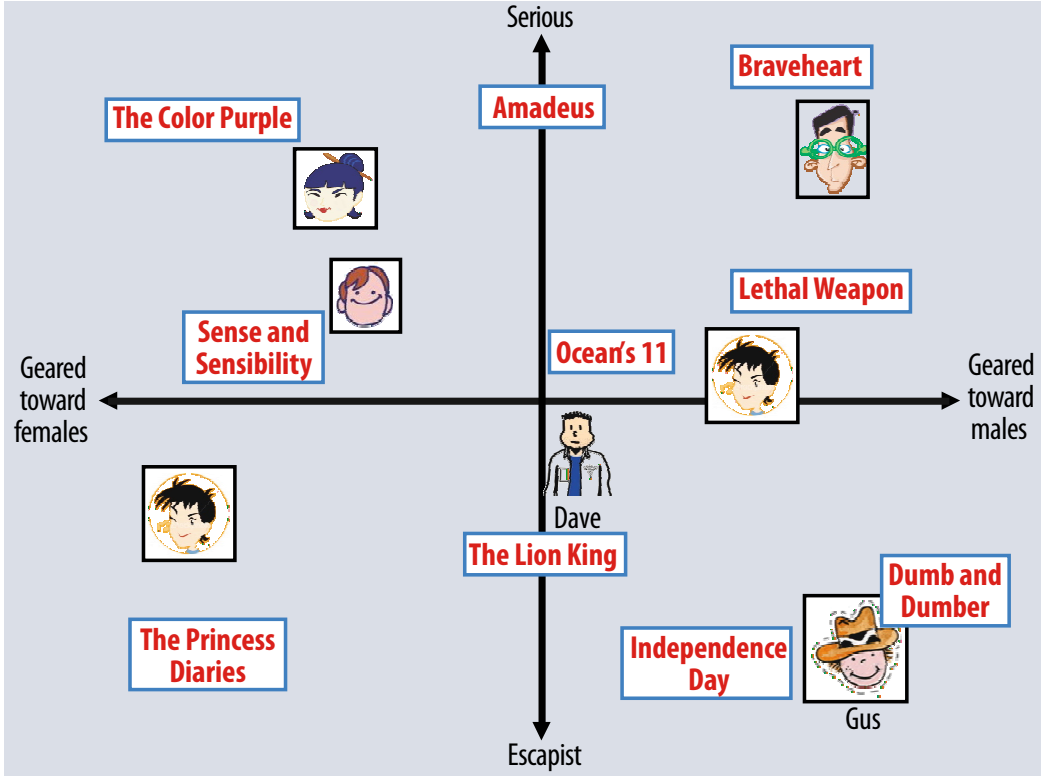
- Neighborhood methods
- Latent factor methods

Neighborhood methods



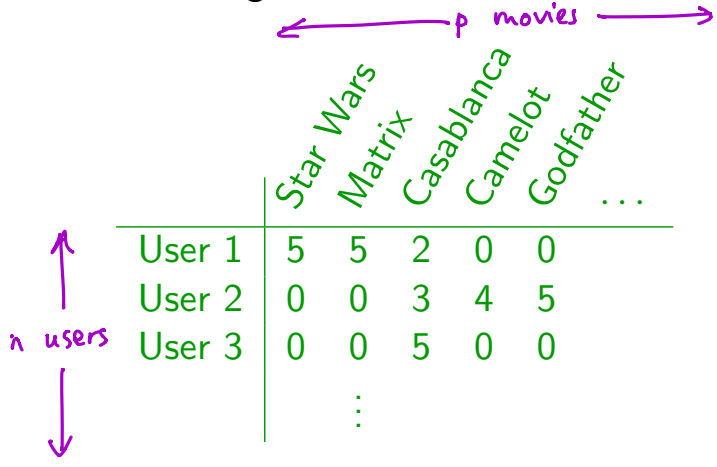
Latent factor methods

Embed all users and all movies into a common low-dimensional space.



The matrix factorization approach

User ratings are assembled in a large matrix M:

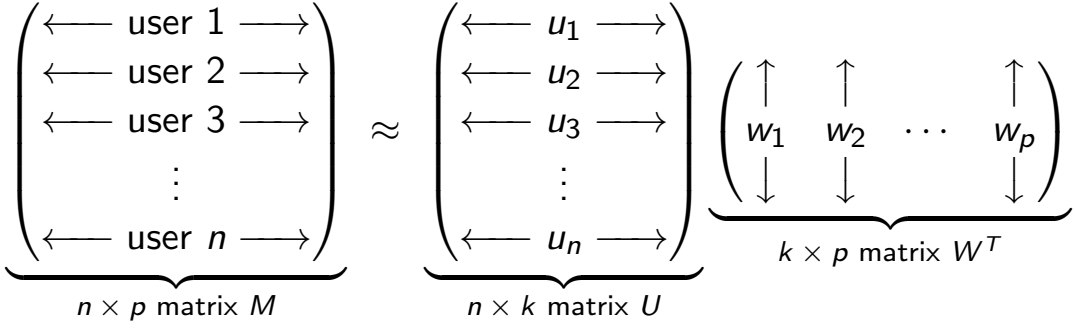


- Not rated = 0, otherwise scores 1-5.
- For n users and p movies, this has size $n \times p$.
- Most of the entries are unavailable, and we'd like to predict these.

Idea: Get best low-rank approximation of M, and use to fill in the missing entries.

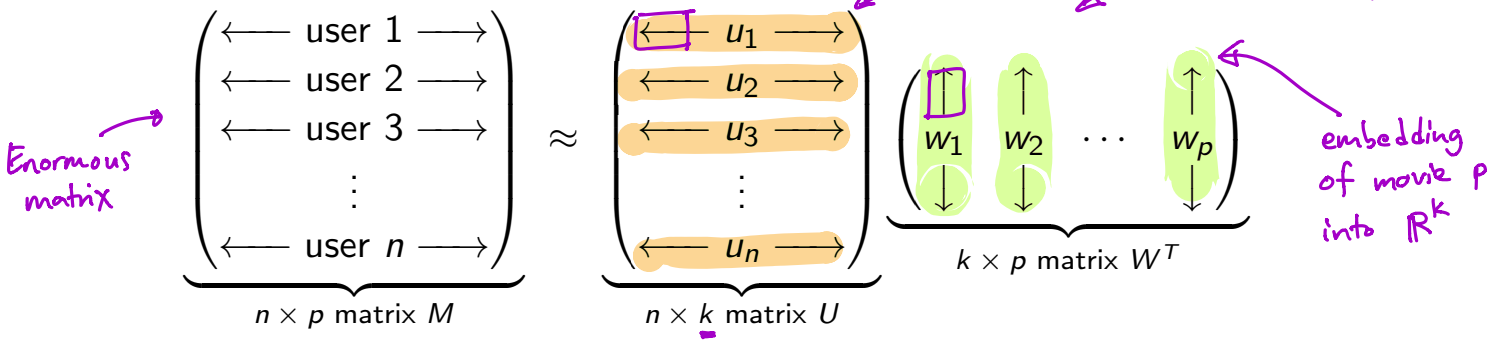
User and movie factors

Best rank-k approximation is of the form $M \approx UW^T$:



User and movie factors

Best rank-k approximation is of the form $M \approx UW^T$:



Thus user i's rating of movie j is approximated as

$$M_{ij} \approx u_i \cdot w_j$$

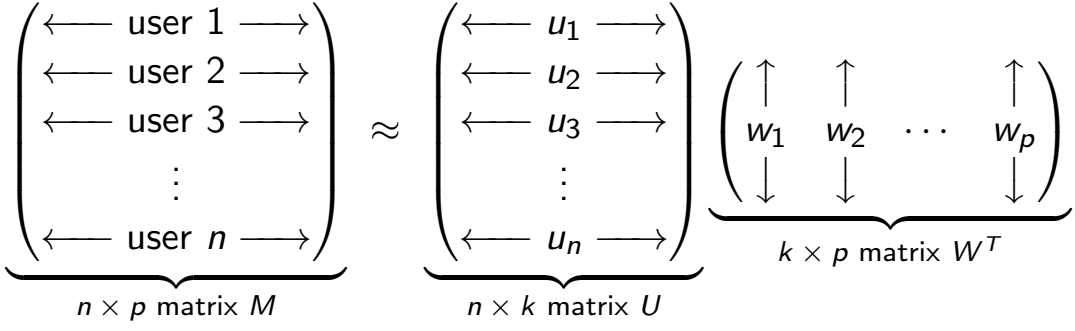
to what extent does user i like movie j?

embedding of user i in $\mathbb{R}^k$

embedding of movie j in $\mathbb{R}^k$

User and movie factors

Best rank-k approximation is of the form $M \approx UW^T$:



Thus user i's rating of movie j is approximated as

$$M_{ij} \approx u_i \cdot w_j$$

"Latent" representation embeds users and movies in the same k-dimensional space:

- Represent ith user by $u_i \in \mathbb{R}^k$
- Represent jth movie by $w_j \in \mathbb{R}^k$

Top two Netflix factors