Recommender Systems

Regina Barzilay
MIT CSAIL

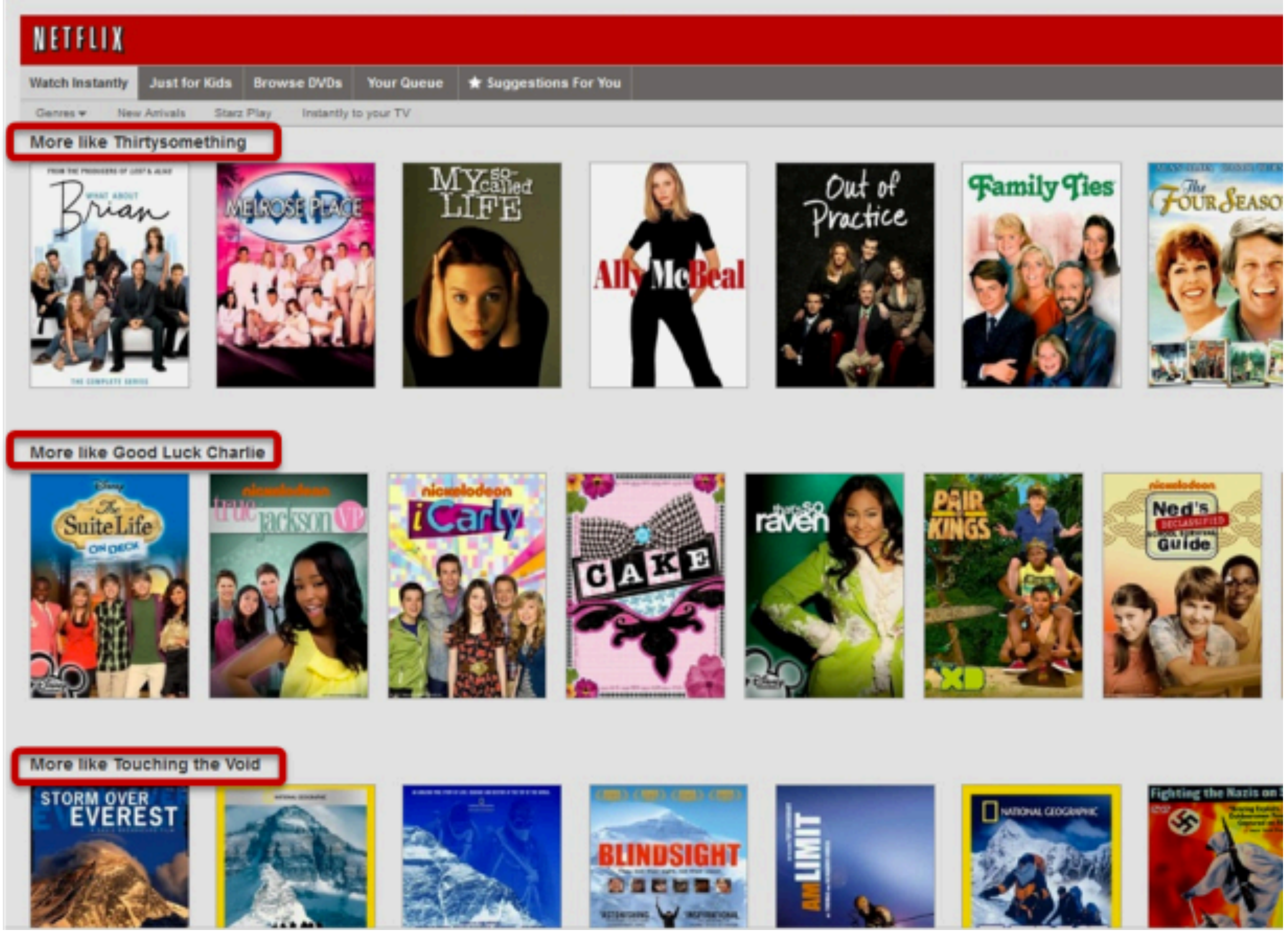
The Netflix Prize

- Between 2006-2009, Netflix held an open competition to improve over their baseline collaborative filtering algorithm. $1M grand prize (for +10%)!

- Training data was given as tuples of <user, movie, rating>.

- Task: predict <user, movie, ?> for unobserved pairs (and with no other features!).

- In general, recommender systems are ubiquitous: Facebook (friend suggestions), Amazon (shopping), Spotify (music discovery), etc.



Problem Definition

- We are given a large, sparse matrix.
- Netflix challenge: $n = 500k$, $m = 18k$, $< 2\,\%$ populated! (Avg. ratings per movie $\sim 5k$).
- Our goal is to compute the missing entries (i.e., matrix completion).
- For a user a and a movie i, we can then predict the "hypothetical" rating, $Y_{a,i}$.

m movies

n users

5	5						5		
		3	5	1	3	4	4		4
	4	2			2				
		5							5
4	5							4	
4							4		
5		4	5	1		4		?	
	4								
5				4					
5						4			
		5				5		3	

$Y_{a,i}$

Observed matrix Y

Supervised Learning (Regression)

- Conceivably, we can pose this as a supervised learning problem: collect some features $\Phi(a)$, and predict $Y_{a,i} = f(\Phi(a))$ where f minimizes $||Y_{a,i} - f(\Phi(a))||^2$ (MSE).

- Features could be actors, director, if it's a comedy, drama, Oscars…

- Maybe this could work! But…

  ➡ What if we don't know ahead of time what are important features? There are many!

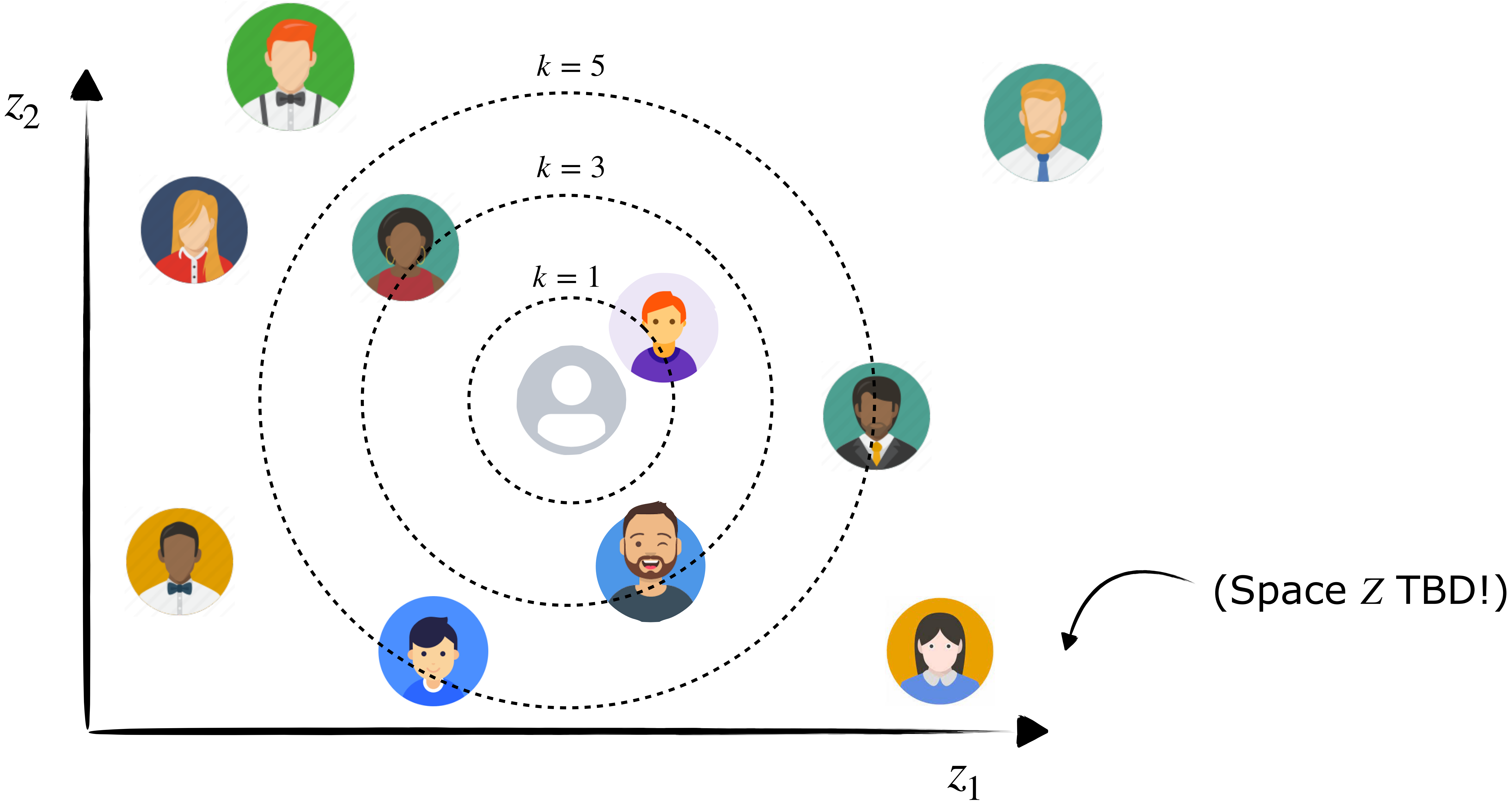
  ➡ What if some features aren't available yet (e.g., Oscar award)?

  ➡ For this to be *personalized*, we need to have enough data for each individual user (a general regressor is no use—e.g., some users love horror, while other users *hate* it!).

- We will explore a different approach: we will learn user *similarities*, with the assumption that under an appropriate similarity metric, similar users behave similarly (if Bob and Alice have similar taste, and Alice likes Titanic, so will Bob).

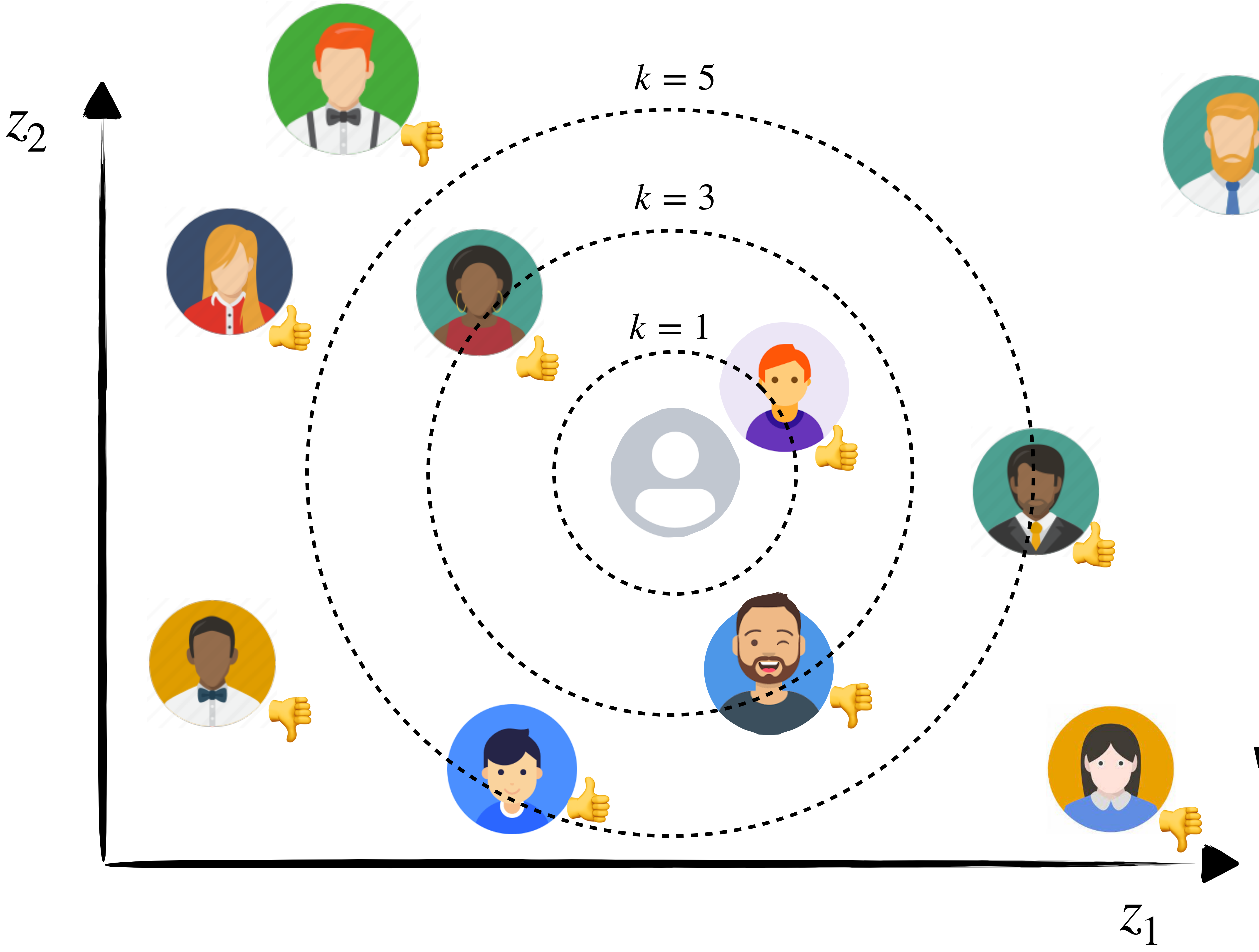
K Nearest Neighbors

- A simple approach: take the average rating of your “friends".



K Nearest Neighbors

- A simple approach: take the average rating of your “friends".



(Space Z TBD!)

K Nearest Neighbors

- A simple approach: take the average rating of your "friends".

$$\hat{Y}_{a,i} = \frac{\sum_{b \in \mathrm{KNN}(a,i)} Y_{b,i}}{K} \qquad \text{where} \quad |\mathrm{KNN}(a,i)| = K$$

- Or a slightly more nuanced variation—*weighted* K nearest neighbors:

$$\hat{Y}_{a,i} = \frac{\sum_{b \in \mathrm{KNN}(a,i)} \mathrm{sim}(a,b) \cdot Y_{b,i}}{\sum_{b \in \mathrm{KNN}(a,i)} \mathrm{sim}(a,b)} \qquad \text{where} \quad |\mathrm{KNN}(a,i)| = K$$

- And many different notions of similarity…

K Nearest Neighbors

- A straightforward way to define similarity between users i and j is via their observed entries (and euclidean distance, hamming distance, etc…).

- Raw entries may not be so good (missing entries, high dim).

user i	5	5						5		
			3	5	1	3	4	4		4
		4	2			2				
			5							5
	4	5							4	
	4							4		
user j	5		4	5	1		4			
		4								
	5				4					
	5						4			
			5				5		3	

Observed matrix Y

K Nearest Neighbors

- A straightforward way to define similarity between users i and j is via their observed entries (and euclidean distance, hamming distance, etc…).
- Raw entries may not be so good (missing entries, high dim).

user i	5	5						5		
			3	5	1	3	4	4		4
		4	2			2				
			5							5
	4	5							4	
	4							4		
user j	5		4	5	1		4			
		4								
	5				4					
	5						4			
user k			5				5		3	

Observed matrix Y

K Nearest Neighbors

- A straightforward way to define similarity between users a and b is via their observed entries (and euclidean distance, hamming distance, etc…).
- Raw entries may not be so good (missing entries, high dim).
- Plenty of other features possible: *normalized score* (how much user a's score deviates from average), side information (gender, age, location…).

user a	5	5						5		
			3	5	1	3	4	4		4
		4	2			2				
			5							5
	4	5							4	
	4							4		
user b	5		4	5	1		4			
		4								
	5				4					
	5						4			
user c			5				5		3	

Observed matrix Y

K Nearest Neighbors: Detractors

- K Nearest Neighbors is a reasonable approach. However, it can still be quite far from the best we can do on this task. Why?

- Issue 1: How you define the similarity is crucial to performance.

- Issue 2: Collecting appropriate features (for the basis of measuring similarity) can be challenging (for the same reasons as for the movies).

- Issue 3: Does not take advantage of the *hidden structure* present in the data.

Collaborative Filtering

- Let's return to our initial problem formulation and objective.

m movies

n users

5	5						5		
		3	5	1	3	4	4		4
	4	2			2				
		5							5
4	5							4	
4							4		
5		4	5	1		4			
	4								
5				4					
5						4			
		5				5		3	

Observed matrix Y

5	5	3	4	1	2	5	5	3	3
2	4	3	5	1	3	4	4	5	4
2	4	2	3	3	2	5	5	3	4
3	3	5	4	2	3	5	5	4	5
4	5	1	5	3	3	4	2	4	5
4	5	2	4	4	2	3	4	5	5
5	4	4	5	1	2	4	3	4	2
5	4	3	4	2	3	3	2	4	2
5	3	5	5	4	4	5	4	3	3
5	4	5	4	1	2	4	4	2	4
4	5	5	4	2	1	5	4	3	4

Completed $\hat{Y}$

A Direct Approach

- Let’s try to solve this direction.

- Objective: let $D = \{(a, i) \colon Y_{a,i}$ is given$\}$ (i.e., our training pairs). Then,

$$\mathcal{J}(\hat{Y}) = \sum_{(a,i)\in\mathcal{D}} \frac{(Y_{a,i} - \hat{Y}_{a,i})^2}{2} + \frac{\lambda}{2}\sum_{a=1}^{n}\sum_{i=1}^{m}\hat{Y}_{a,i}$$

Regularization term

- Great! Let’s take the derivative and set to 0.

A Direct Approach

$$\frac{\partial \mathcal{J}(\hat{Y}_{a,i})}{\partial \hat{Y}_{a,i}} = \frac{\partial}{\partial \hat{Y}_{a,i}} \sum_{(a,i)\in\mathcal{D}} \frac{(Y_{a,i} - \hat{Y}_{a,i})^2}{2} + \frac{\lambda}{2} \sum_{a=1}^{n} \sum_{i=1}^{m} \hat{Y}_{a,i} = 0$$

- Compute this and solve for $\hat{Y}_{a,i}$ (can be done as an exercise):
  - For $(a,i) \in D$: $\hat{Y}_{a,i} = \dfrac{Y_{a,i}}{1+\lambda}$.
  - For $(a,i) \notin D$: $\hat{Y}_{a,i} = 0$.

A Direct Approach

$$\frac{\partial \mathcal{J}(\hat{Y}_{a,i})}{\partial \hat{Y}_{a,i}} = \frac{\partial}{\partial \hat{Y}_{a,i}} \sum_{(a,i)\in\mathcal{D}} \frac{(Y_{a,i} - \hat{Y}_{a,i})^2}{2} + \frac{\lambda}{2}\sum_{a=1}^{n}\sum_{i=1}^{m} \hat{Y}_{a,i} = 0$$

- Compute this and solve for $\hat{Y}_{a,i}$:
  - For $(a,i) \in D$: $\hat{Y}_{a,i} = \dfrac{Y_{a,i}}{1+\lambda}$.
  - For $(a,i) \notin D$: $\hat{Y}_{a,i} = 0$.

What? This is worse than just taking $Y_{a,i}$!

What Went Wrong?

- We didn't leverage any similarities or structure in our data.
- We fit $m \times n$ parameters (huge!) independently.
- There is no connection between the assignment of values for $\hat{Y}_{a,i}$ and $\hat{Y}_{b,j}$.
- No clusters are discovered, no dependencies exploited… we simply repeat the same matrix (but distorted to be smaller, if using regularization).
- Let's address this by making a few assumptions.

Matrix Factorization

- Assumption: Y is low rank.
- In other words, rows (or columns) of Y are not linearly independent.
- Example:

$$\underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 5 & 10 & 15 \end{bmatrix}}_{\hat{Y}} = \underbrace{\begin{bmatrix} 1 \\ 5 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}}_{V}$$

- In our case, we can compute $Y \approx \hat{Y} = UV^\top$, where U and V are low rank matrices.

Matrix Factorization: Objective

- Objective when using a low rank approximation:

$$\mathcal{J}(\hat{Y}) = \sum_{(a,i)\in\mathcal{D}} \frac{(Y_{a,i} - [UV^\top]_{a,i})^2}{2} + \frac{\lambda}{2}\left(\sum_{a=1}^{n}\sum_{j=1}^{d} U_{a,j}^2 + \sum_{i=1}^{m}\sum_{j=1}^{d} V_{i,j}^2\right)$$

- To optimize U and V, we apply alternating least squares: fix U and optimize V, then fix V and optimize U, and repeat until convergence.

- This is a common optimization strategy in machine learning (e.g., EM, coordinate descent, etc).

- SGD would work too.

Matrix Factorization: Worked Example

Relation to SVD

- Our matrix factorization approach is very similar to singular value decomposition (SVD), where for any $m \times n$ matrix, we can write

$$Y = U\Sigma V^*$$

- Σ is a diagonal matrix of eigenvalues, while U and V form orthonormal bases for $\mathbb{R}^n$ and $\mathbb{R}^m$.

- If we truncate Σ to the top k eigenvalues, then U and V will be of rank k, and will yield the best reconstruction of Y (without any regularization).

- This approach can be solved exactly using linear algebra (but is slow), and has been applied (well before, e.g., Billsus & Pazzani, 1998) to recommender systems, but with less practical success (perhaps due to regularization).

Matrix Factorization++

- In most recommender problems, there is a selection bias in the observed data.

- For example, users don't rate movies at random, and they may not even rate them at all (so, wasted data?).

- But! They *pick* movies to watch in the first place (which indicates preference).

- A simple trick: add a bias term for each movie that is adjusted by the number of total movies that user a provided an implicit preference for.

- In other words, there is a "prior"-like assumption that states that a user is more likely to "like" a movie they rate, over a random, not-rated movie.

- See *Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model* (Koren, 2008).

Learning Representations

- Matrix factorization is an example of *representation learning*.

- Rows of U and V are low-dimensional vectors mapping users and movies to a shared subspace in $\mathbb{R}^d$ (where dot products recover ratings).

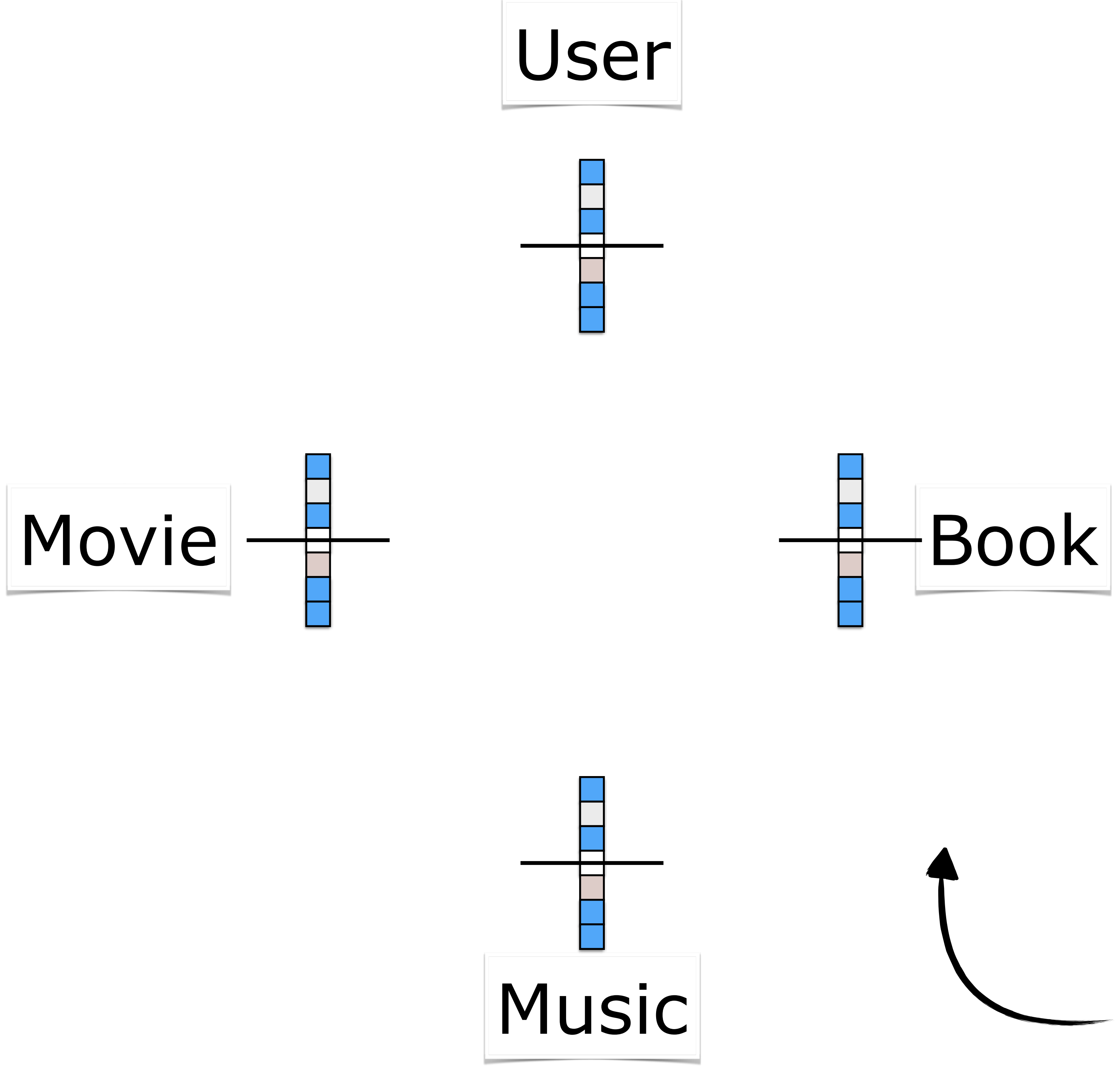
- We can be far more general:

  - Nonlinear relationships: U and V are low-rank matrices, but $g(U)h(V)^\top$ recovers Y, for some non-linear functions g and h.

  - Contrastive learning: we don't recover an observation matrix Y, but enforce that vectors U_a and V_b should be closer to each other than U_c and V_b (or U_a and V_c).

  - Collaborative bags: F and G are low-rank matrix of *components*, U and V are formed from bags of components (related by maps A and B), i.e., $U_a = \sum_{c \in A_a} F_c$, $V_b = \sum_{c \in B_b} G_c$.

Learning Representations

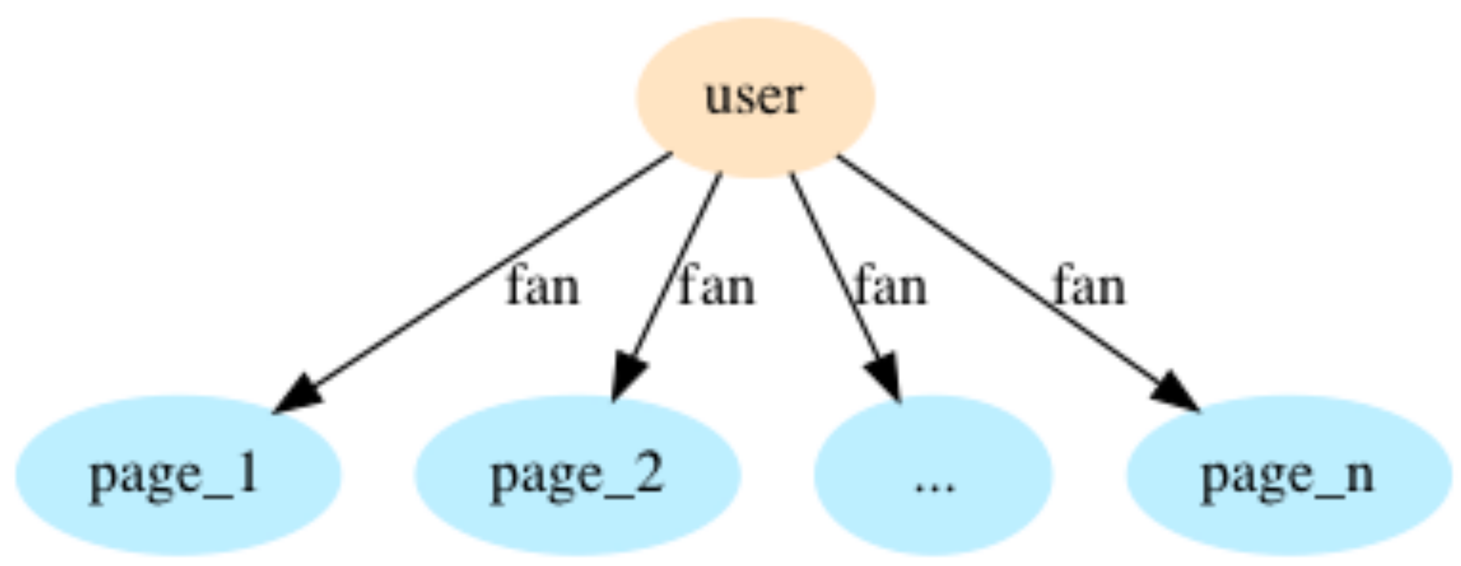


All embedded in $\mathbb{R}^d$!

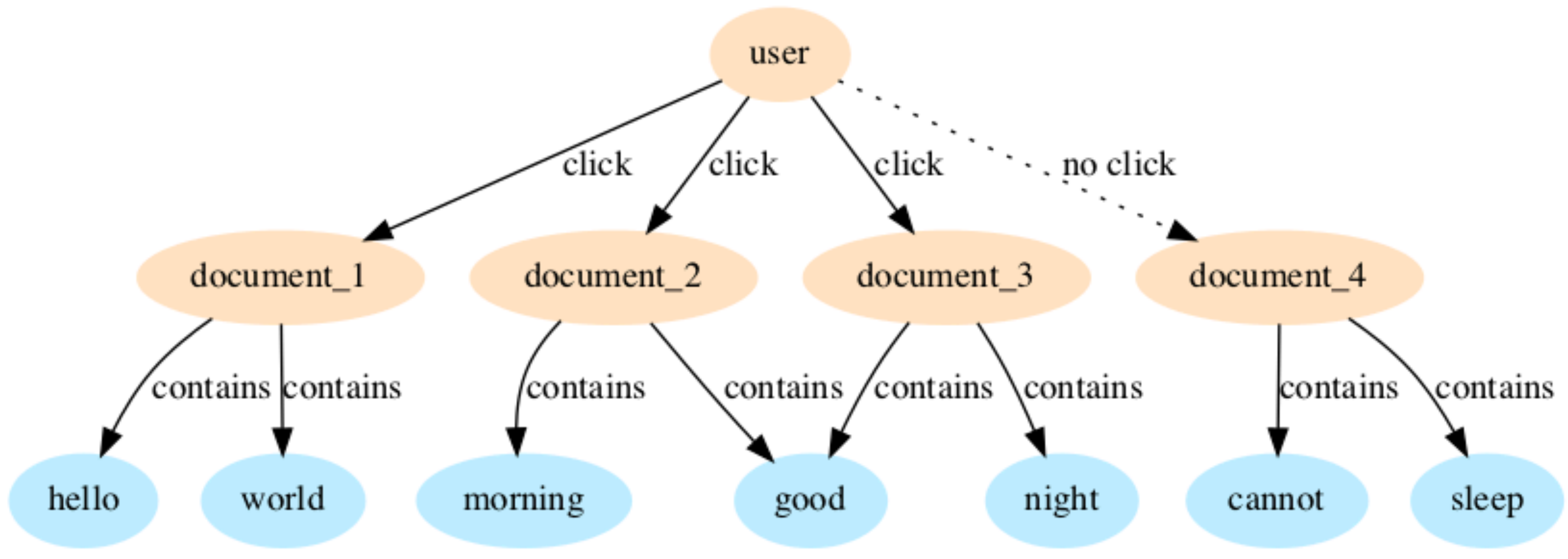
StarSpace: Embed All The Things!

Wu, Fisch, Chopra, Adams, Bordes, Weston (2018)

- General purpose embedding system for link prediction, word or document level embeddings, information retrieval, graph embeddings, image classification, etc.
- Builds representations from collections of low rank (learned) matrices!
- Objective is something you can define: i.e., image to caption, images of the same class, documents of the same topics, users that like the same pages, etc.



FB Page recommendation



Article recommendation

Summary

- Collaborative filtering takes advantage of latent shared structure.
- Key idea: observations are related! Hence, observational matrix is low rank.
- Matrix factorization techniques can find low rank approximations effectively for partially observed matrices; the missing entries can then be recovered.
- More generally, MF is a flexible framework that can be extended to more abstract *representation* learning, where objects are given a low dimensional featurization (that is learned through data via some structural objective).
- More on learning lexical representations tomorrow!