Recommendation Systems
Part II

Devavrat Shah

Massachusetts Institute of Technology

Recall — Part I

Introduction, simple methods

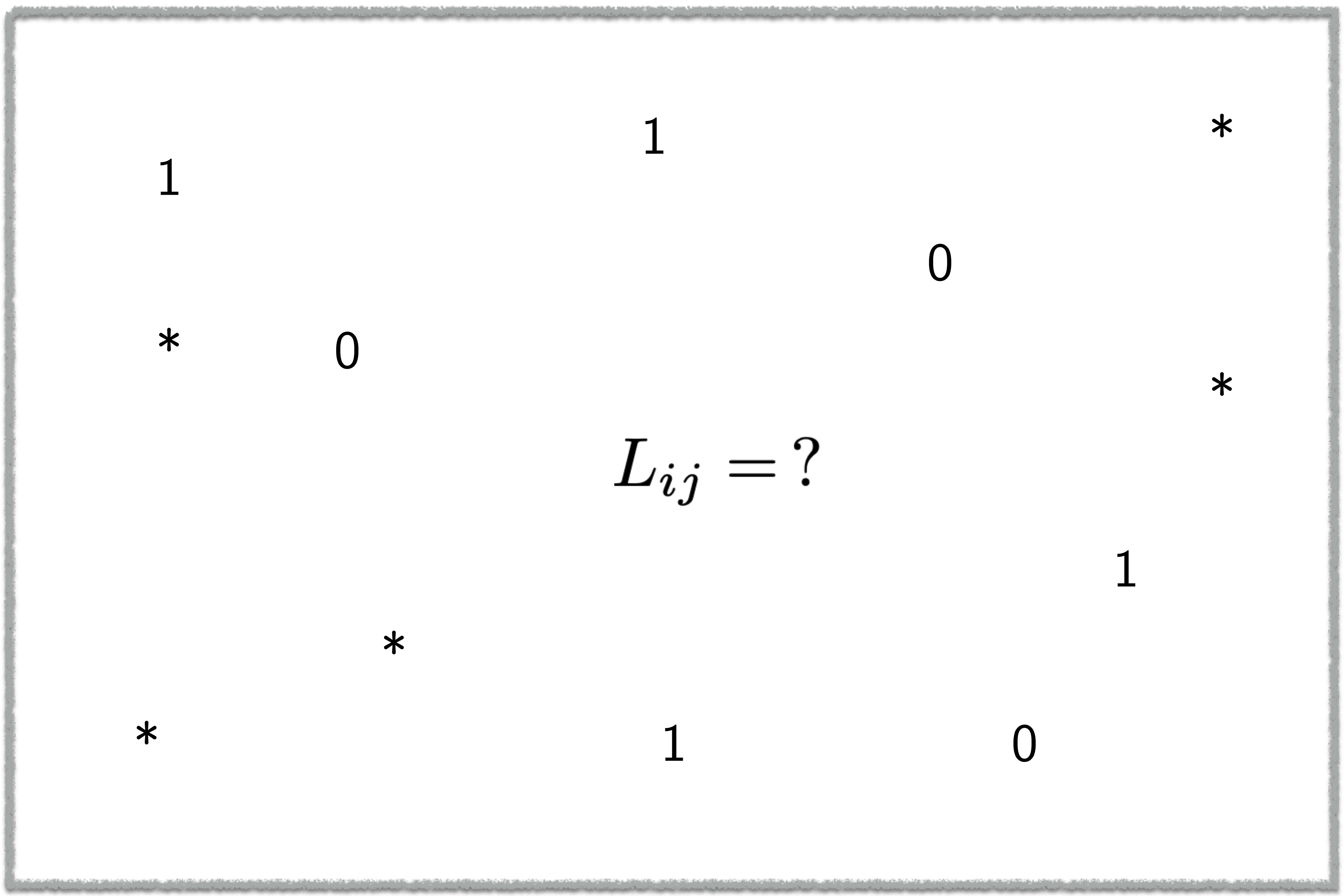
- Module 1: background
  - Recommendation systems: why and what?
  - Example datasets
- Module 2: problem statement
  - Recommendation systems: a prediction problem
  - Model: estimating time varying tensor with side information
- Module 3: simple solutions
  - Solution I: averaging
  - Solution II: content-based

Recall: problem statement

We will start with simple problem statement: complete the matrix

j

i



users

items / providers

Recommendation: Problem statement

We will start with simple problem statement: complete the matrix

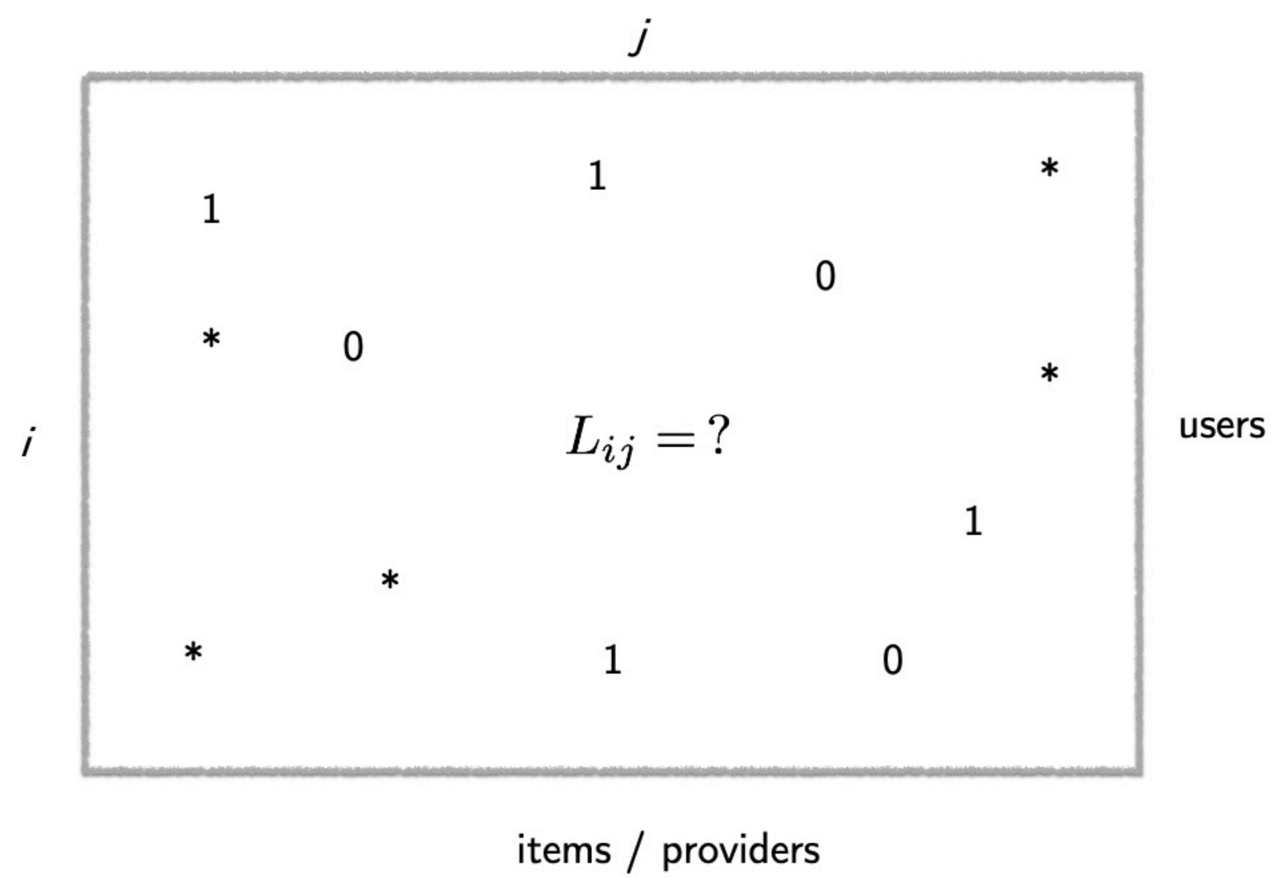
Observations: Y_{ij} over i in users, j in items

If (i, j) is observed

$$E\left[Y_{ij}\right] = L_{ij}$$

If (i, j) is *not* observed

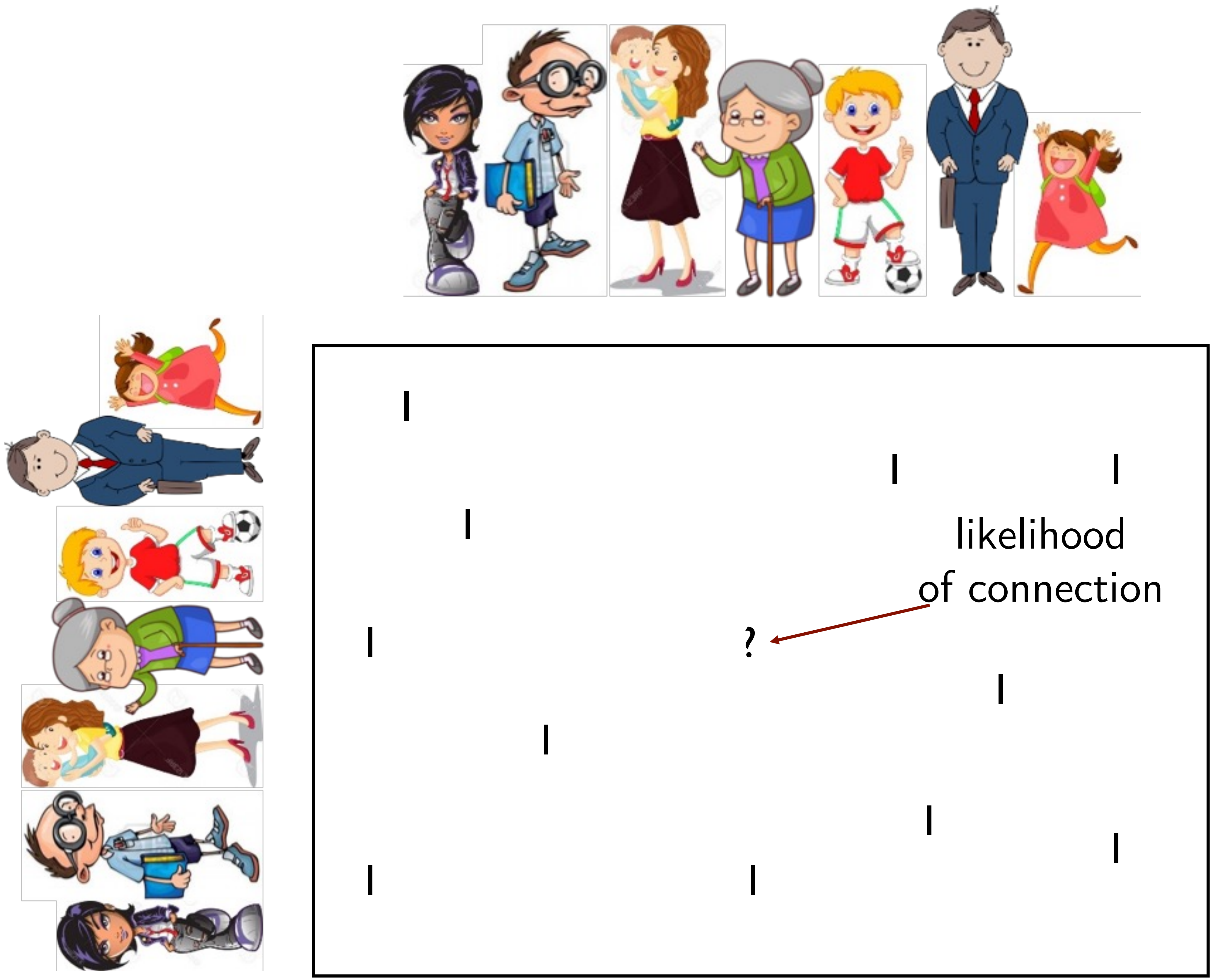
$$Y_{ij} = \star \text{ or } ?$$



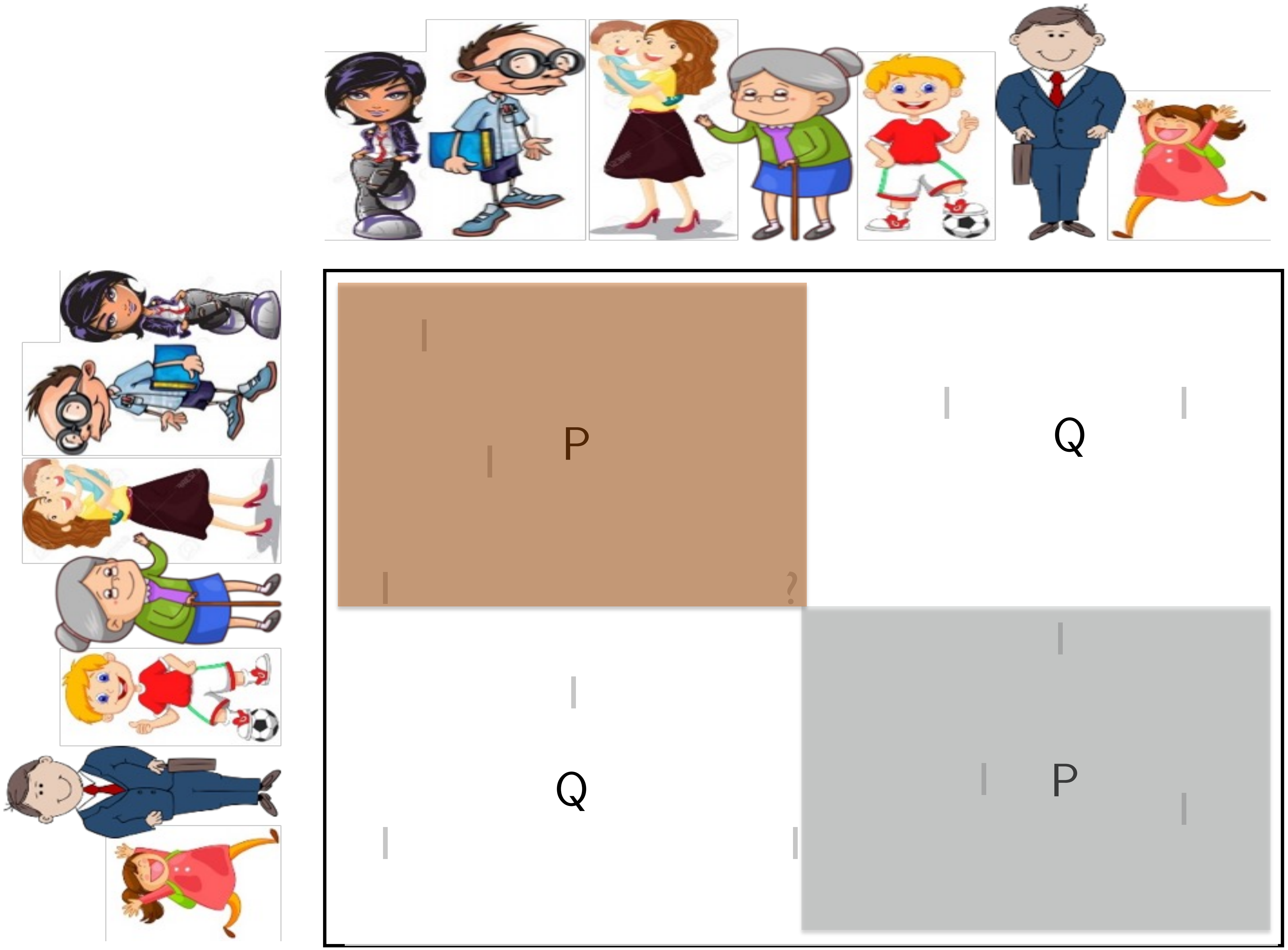
Goal: produce estimation $\hat{L}_{ij}$ for all i, j

so that $\hat{L}_{ij} \approx L_{ij}$ for all i, j

Social Networks

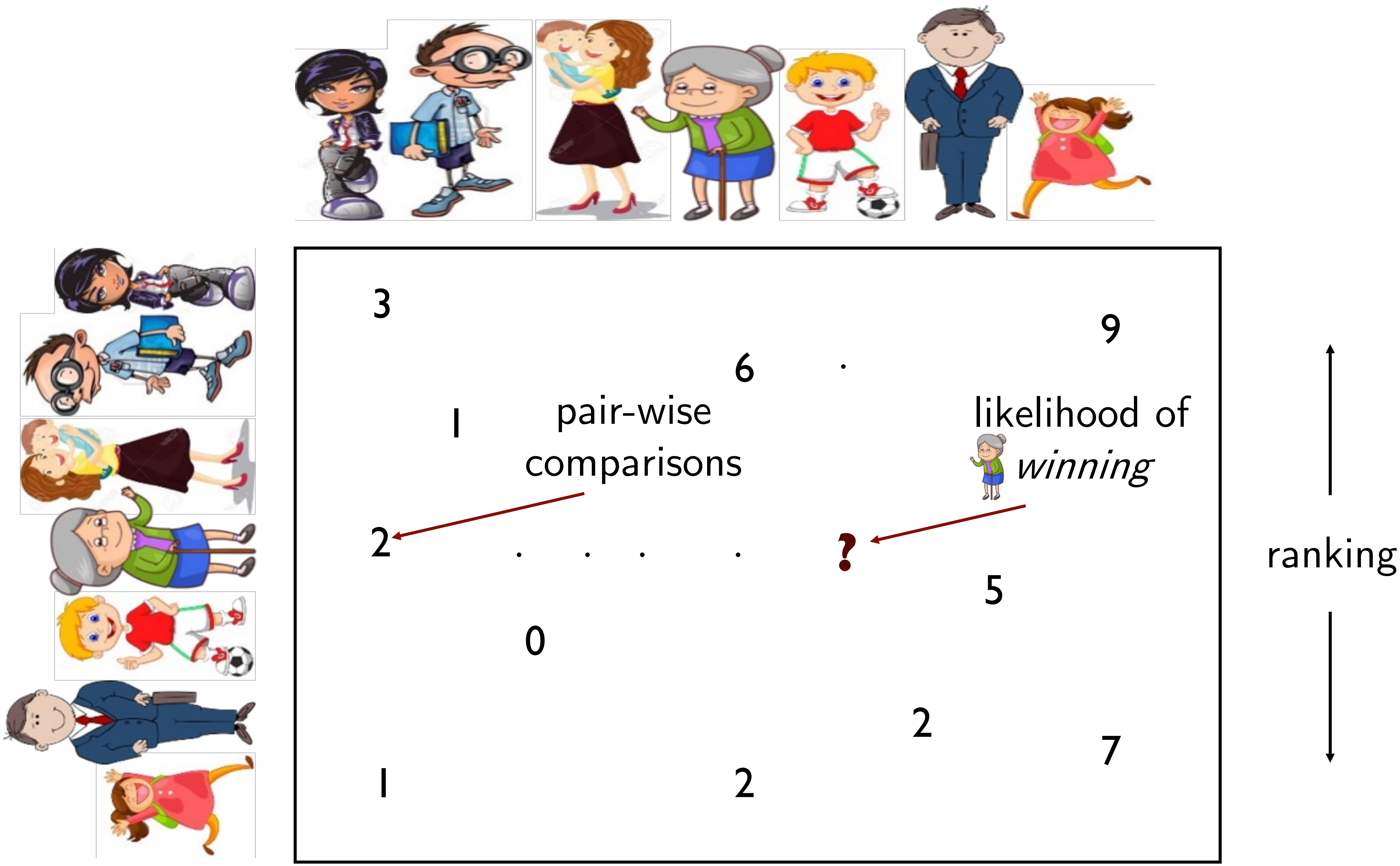


Community Detection



$P > Q$

Ranking Players, Teams



Crowd-sourcing

Is the Website Suitable for Children?

Recommendations

May be *data* can help. What data?

Example: Yelp data

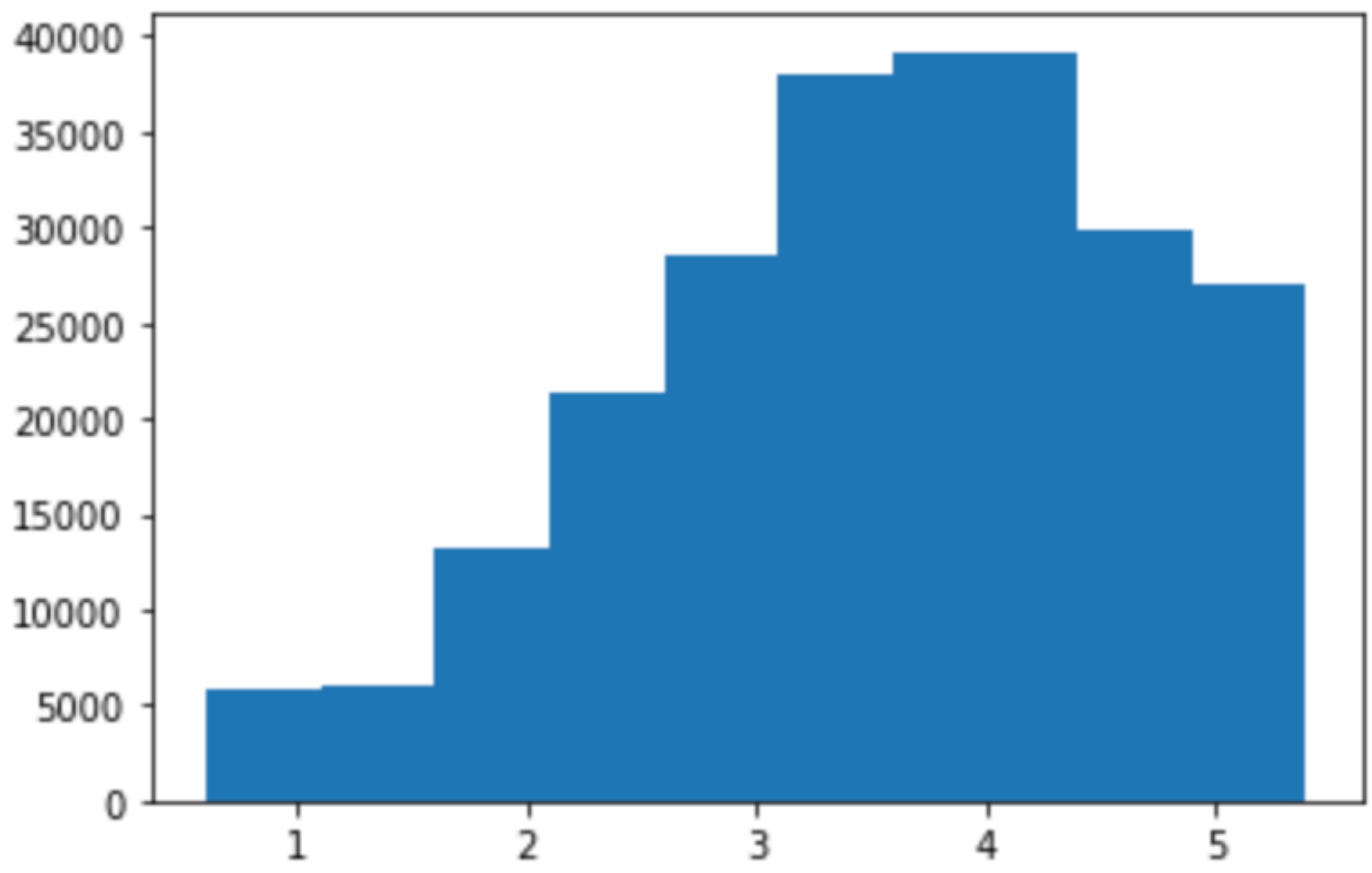
- Businesses: attributes (locations, category), hours
- Users: attributes, friends
- Reviews: rating, description, time
- Check-ins: time
- Tip

Exercise: go to link below, explore data and reproduce statistics reported

https://www.kaggle.com/jagangupta/what-s-in-a-review-yelp-ratings-eda

Explore: Yelp Data

(Aggregate) Star Rating Distribution



Data URL:

https://www.kaggle.com/jagangupta/what-s-in-a-review-yelp-ratings-eda

Explore: Yelp Data

What fraction of reviews are known?

Users = ~2M

Businesses = ~200k

Total possible reviews = ~2M x ~200k = ~0.4T

Known reviews = ~8M

Fraction known = ~8M / 0.4 T = 2 x 10^{-5}

i.e. 2 in every 100k reviews is known, rest are *unknown*

Finding these *unknown* reviews is the primary goal of Rec Sys

Data URL:

https://www.kaggle.com/jagangupta/what-s-in-a-review-yelp-ratings-eda

Explore: MovieLens Data

MovieLens Data

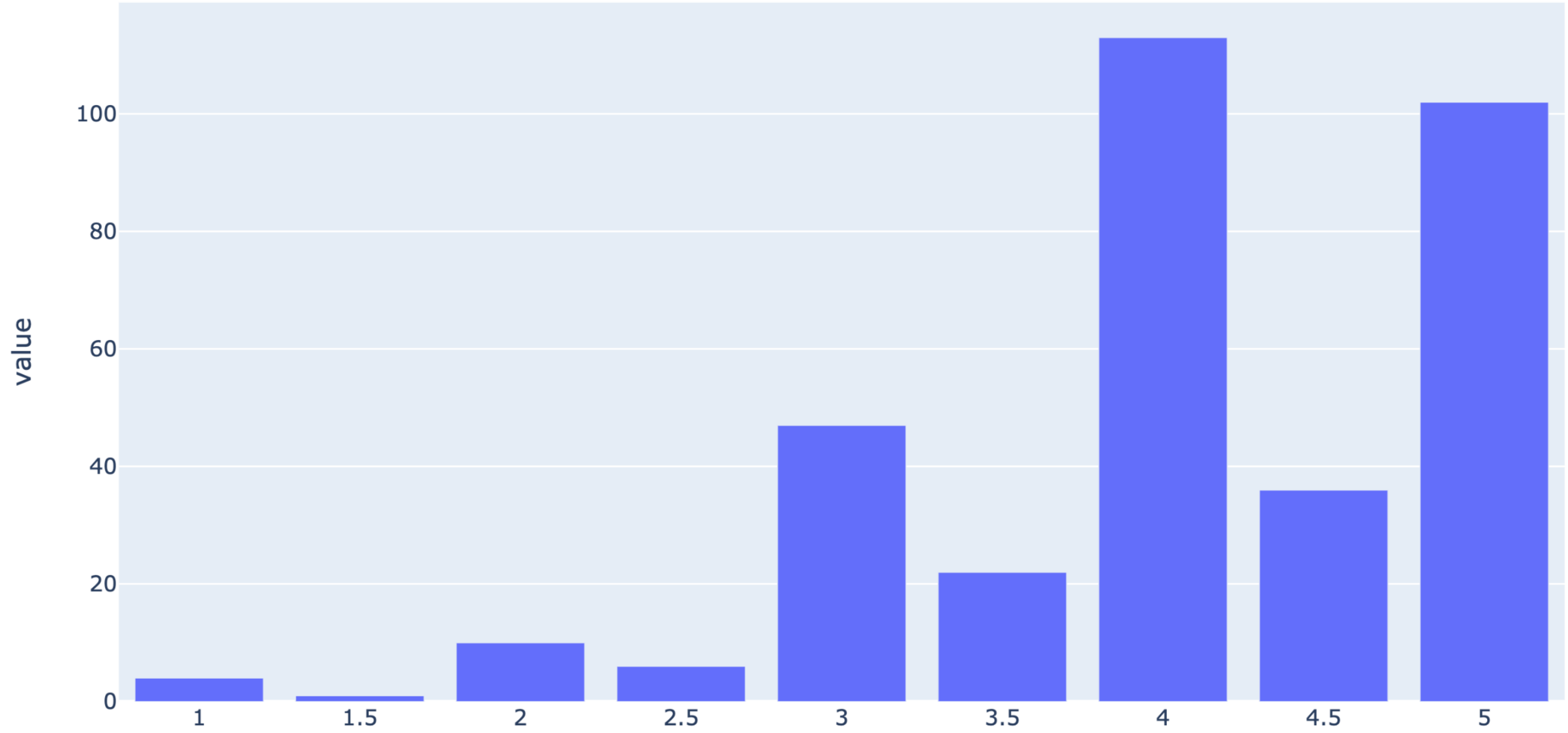
- Movies: attributes including title, release date, genre, actors, director
- Users: demographics including age, gender, occupation, zip code
- Reviews: ratings, timestamp

Exercise: go to link below, explore data and reproduce statistics reported

https://grouplens.org/datasets/movielens/100k

Explore: MovieLens Data

Distribution of top-rated movie (356)



Data URL:

https://grouplens.org/datasets/movielens/100k

Explore: MovieLens Data

What fraction of reviews are known?

Users = ~1.7k

Movies = ~1k

Total possible reviews = ~1.7k x ~1k = ~1.7M

Known reviews = ~100k

Fraction known = ~100k / 1.7M = ~0.058 or ~6%

i.e. 6 in every 100 reviews is known, rest are *unknown*

Finding these *unknown* reviews is the primary goal of Rec Sys

Data URL:

https://www.kaggle.com/jagangupta/what-s-in-a-review-yelp-ratings-eda

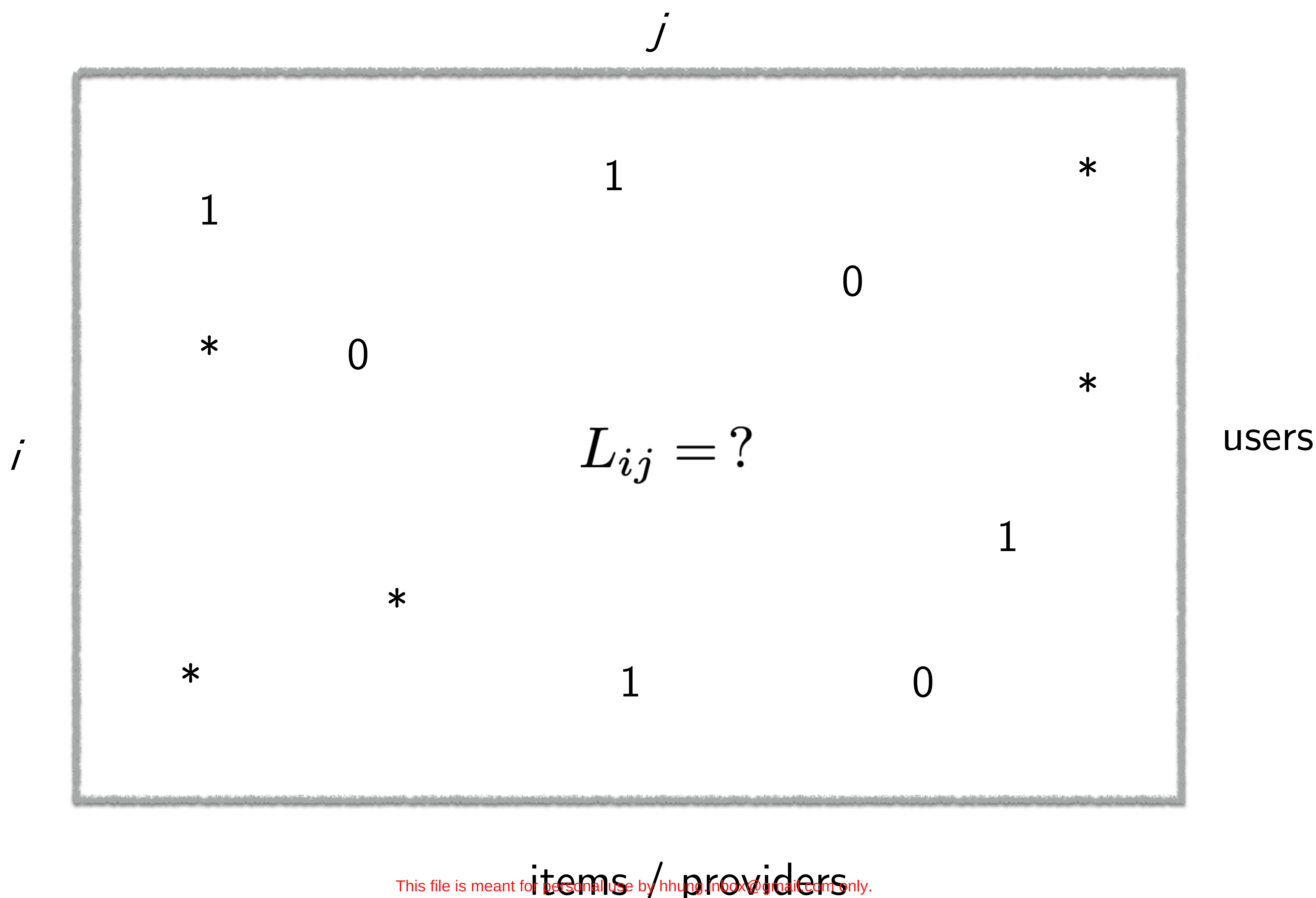
Recall — Part I

Introduction, simple methods

- Module 1: background
  - Recommendation systems: why and what?
  - Example datasets
- Module 2: problem statement
  - Recommendation systems: a prediction problem
  - Model: estimating time varying tensor with side information
- Module 3: simple solutions
  - Solution I: averaging
  - Solution II: content-based

Recall: problem statement

We will start with simple problem statement: complete the matrix



Recommendation: Problem statement

Complete the matrix

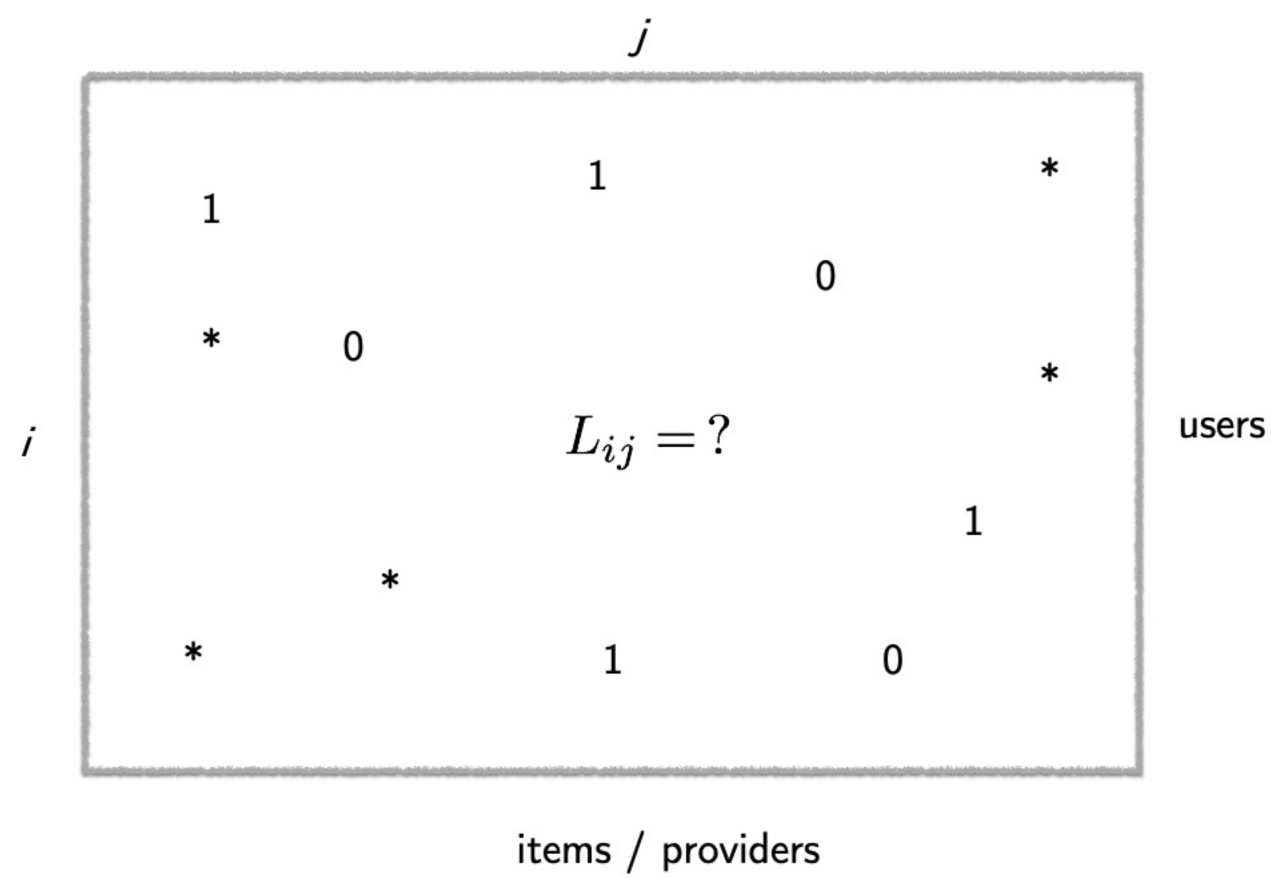
Observations: Y_{ij} over i in users, j in items

If (i, j) is observed

$$E[Y_{ij}] = L_{ij}$$

If (i, j) is *not* observed

$$Y_{ij} = \star \text{ or } ?$$

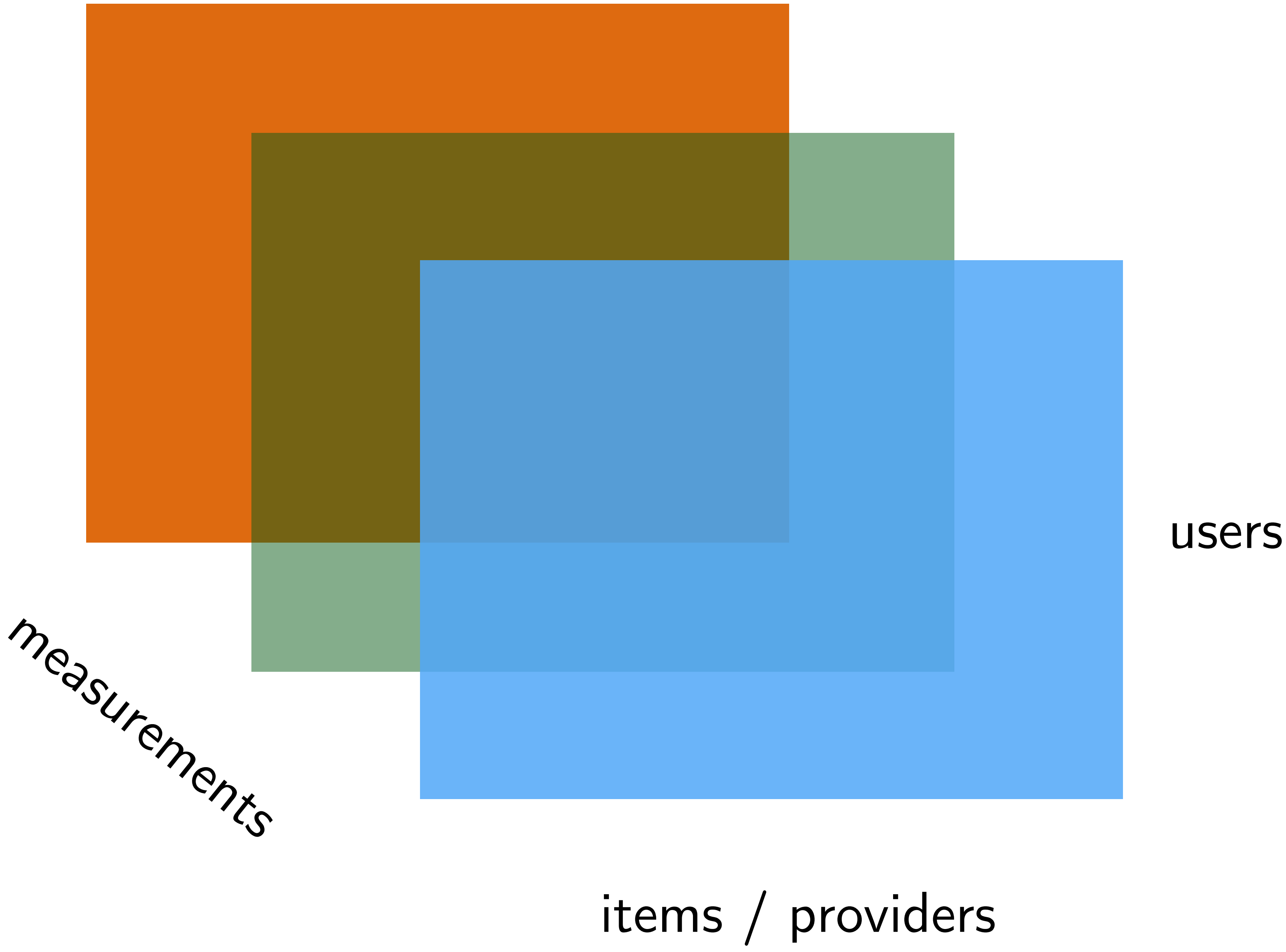


Goal: produce estimation $\hat{L}_{ij}$ for all i, j

so that $\hat{L}_{ij} \approx L_{ij}$ for all i, j

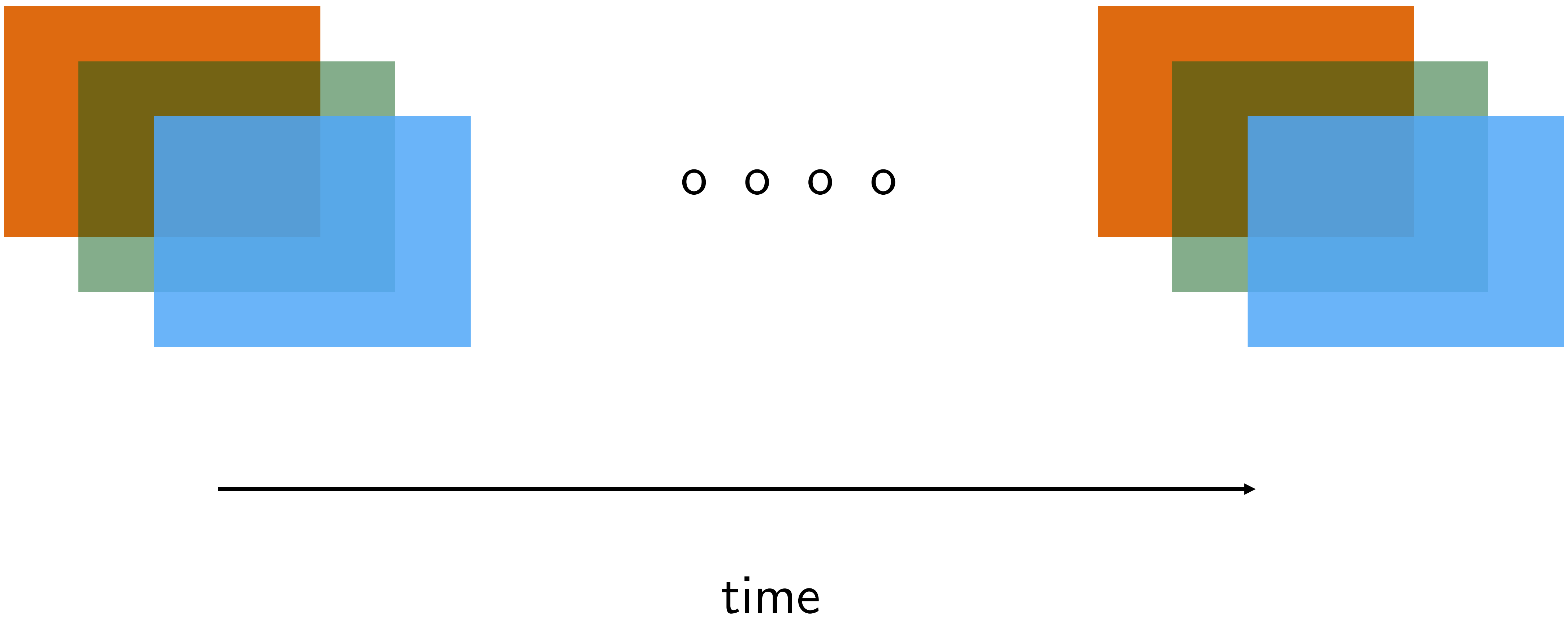
Recommendation: Problem statement

Prediction problem: complete the tensor



Recommendation: Problem statement

Prediction problem: complete the time varying tensor



Recall — Part I

Introduction, simple methods

- Module 1: background
  - Recommendation systems: why and what?
  - Example datasets
- Module 2: problem statement
  - Recommendation systems: a prediction problem
  - Model: estimating time varying tensor with side information
- Module 3: simple solutions
  - Solution I: averaging
  - Solution II: content-based

Solution 1: Averaging

What if, instead we assume

All items or providers are identical

Then estimate: row average (+ correction for number of observations)

How to put these two simple estimators together?

$$2L_{ij} = L_{i\cdot} + \frac{1}{\sqrt{n_{i\cdot}}} + L_{\cdot j} + \frac{1}{\sqrt{n_{\cdot j}}}$$

where $L_{i\cdot}$ is average of observed entries in row i

$n_{i\cdot}$ is number of observed entries in row i

$L_{\cdot j}$ is average of observed entries in column j

$n_{\cdot j}$ is number of observed entries in column j

Solution 2: Content Based

This is *supervised learning* problem we have already seen

Labeled data:

each observed entry in matrix (i, j) corresponds to labeled data

$$((x_i, y_j); L_{ij})$$

Learning problem:

learn the model / function that maps features to label

For likelihood setting with observations being 0 or 1, it is *classification*

Exercise:

What method would you use for classification?

What if observations were not 0/1 but continuous numbers?

Appendix: Converting Content to Features

Challenge: Content is *not* structured

e.g. recall user information from MovieLens data

```
user id | age | gender | occupation | zip code
```

```
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
```

How do we convert these "attributes" or "content" to features

Appendix: Converting Content to Features

Challenge: Content is *not* structured

How do we convert these "attributes" or "content" to features

Age: It's a number. That's easy.

Gender: Two classes or binary. Convert into 0 / 1.

Occupation:

Treat as a class. Use one-hot encoding.

```
user id | age | gender | occupation | zip code


1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
```

Appendix: Converting Content to Features

Challenge: Content is *not* structured

What about "Tip" data. It has free-form text.

	business_id	compliment_count	date	text	user_id
0	UYX5zL_Xj9WEc_Wp-FrqHw	0	2013-11-26 18:20:08	Here for a quick mtg	hf27xTME3EiCp6NL6VtWZQ
1	Ch3HkwQYv1YKw_FO06vBWA	0	2014-06-15 22:26:45	Cucumber strawberry refresher	uEvusDwoSymbJJ0auR3muQ
2	rDoT-MgxGRiYqCmi0bG10g	0	2016-07-18 22:03:42	Very nice good service good food	AY-laIws3S7YXNl_f_D6rQ
3	OHXnDV01gLokiX1ELaQufA	0	2014-06-06 01:10:34	It's a small place. The staff is friendly.	Ue_7yUlkEbX4AhnYdUfL7g
4	GMrwDXRlAZU2zj5nH6l4vQ	0	2011-04-08 18:12:01	8 sandwiches, $24 total...what a bargain!!! An...	LltbT_fUMqZ-ZJP-vJ84IQ

Need an approach to convert text into number or vector of numbers

Appendix: Converting Content to Features

Text to vector of number:

Create word-frequency in documents matrix M

words

Here Cucumber Sandwich j o o o

documents

1

2

i

o

o

o

word j frequency
in doc i

= M

Appendix: Converting Content to Features

Text to vector of number:

Create word-frequency in documents matrix M

Perform Principal Component Analysis of M

Each document receives k co-ordinates

via k principal components

This is the vector representing the text features (restricted to data)

Another (more classical) option:

TF-IDF vector

But it can be very large

Outline — Part II

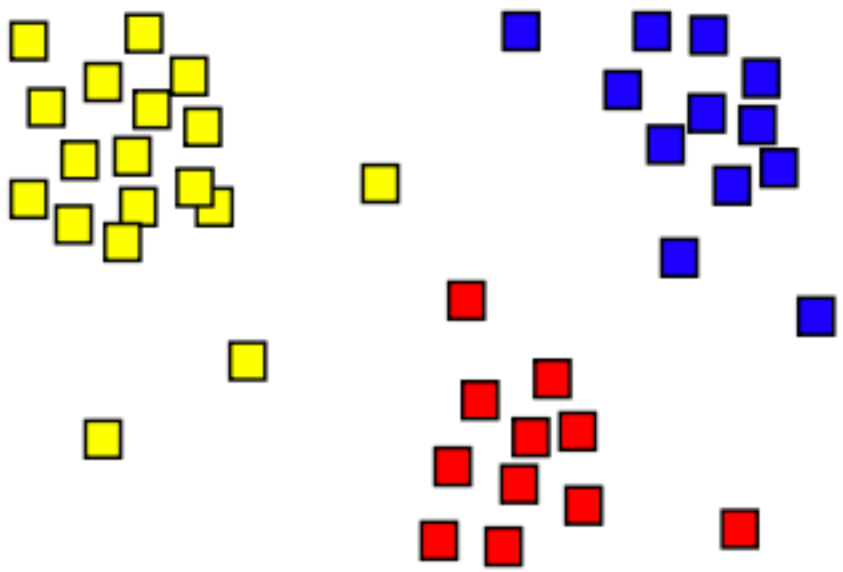
Solution evolves, Matrix estimation

- Module 1: clustering
  - Finding user and item clusters
  - Averaging *within* clusters
- Module 2: collaborative filtering aka personalized clustering
  - Finding users and items similar to a given user, item
  - Averaging *amongst* user-item specific *similar* users, items
- Module 3: singular value thresholding, optimization
  - It's a Matrix! find Singular Value Decomposition
  - Solve for least-squares

Module 1: clustering

Clustering

What is a cluster?



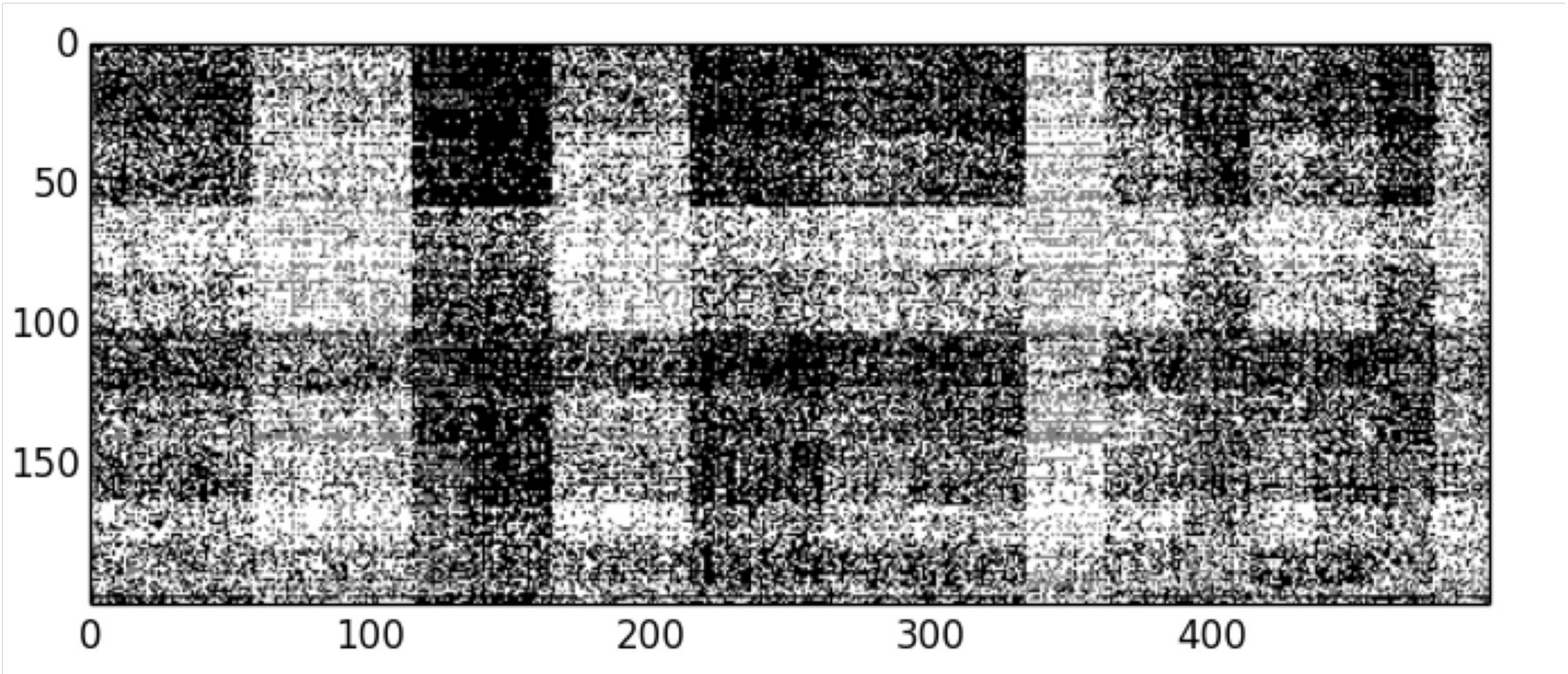
An example of Three clusters

Loosely speaking, a *cluster* is a collection of observations

that are *more* similar to each other than the *rest*

Clustering: MovieLens

Visual representation (after re-ordering) of rating matrix



top 200 users x top 500 items

Clustering

We are interested in clusters of *users* and *items*

Why?

To estimate L_{ij} , we compute average over *all* rows, columns

This assumes *all* users (or items) being *homogenous*

Most likely that may not be true

However, it may be that users (and items) form

multiple homogenous enough groups or clusters

Then find these clusters and

restrict averaging method to the cluster

in which user/item of interest belongs

Clustering

How to cluster users (or items)?

Step 1. Compute similarity between each pair of N users [How?]

This gives N x N similarity matrix (that is symmetric)

Step 2. Obtain representation of each users in low-dim space [How?]

This assigns co-ordinates to each user in d dim space

Step 3. Perform k-means clusterings [How?]

Iteratively find k clusters till they make sense

Explore: Yelp Data

Exercise:

Apply clustering algorithm for Yelp Data

Compare the estimation error with respect to global averaging

Did it work better?

Data URL:

https://www.kaggle.com/jagangupta/what-s-in-a-review-yelp-ratings-eda

Module 2: collaborative filtering

Personalized Clustering

Aggregate clustering

leads to many user (or item) being closer to 'boundary' of clusters

that is the cluster utilized for its estimation

may not be representative enough

Personalized clustering

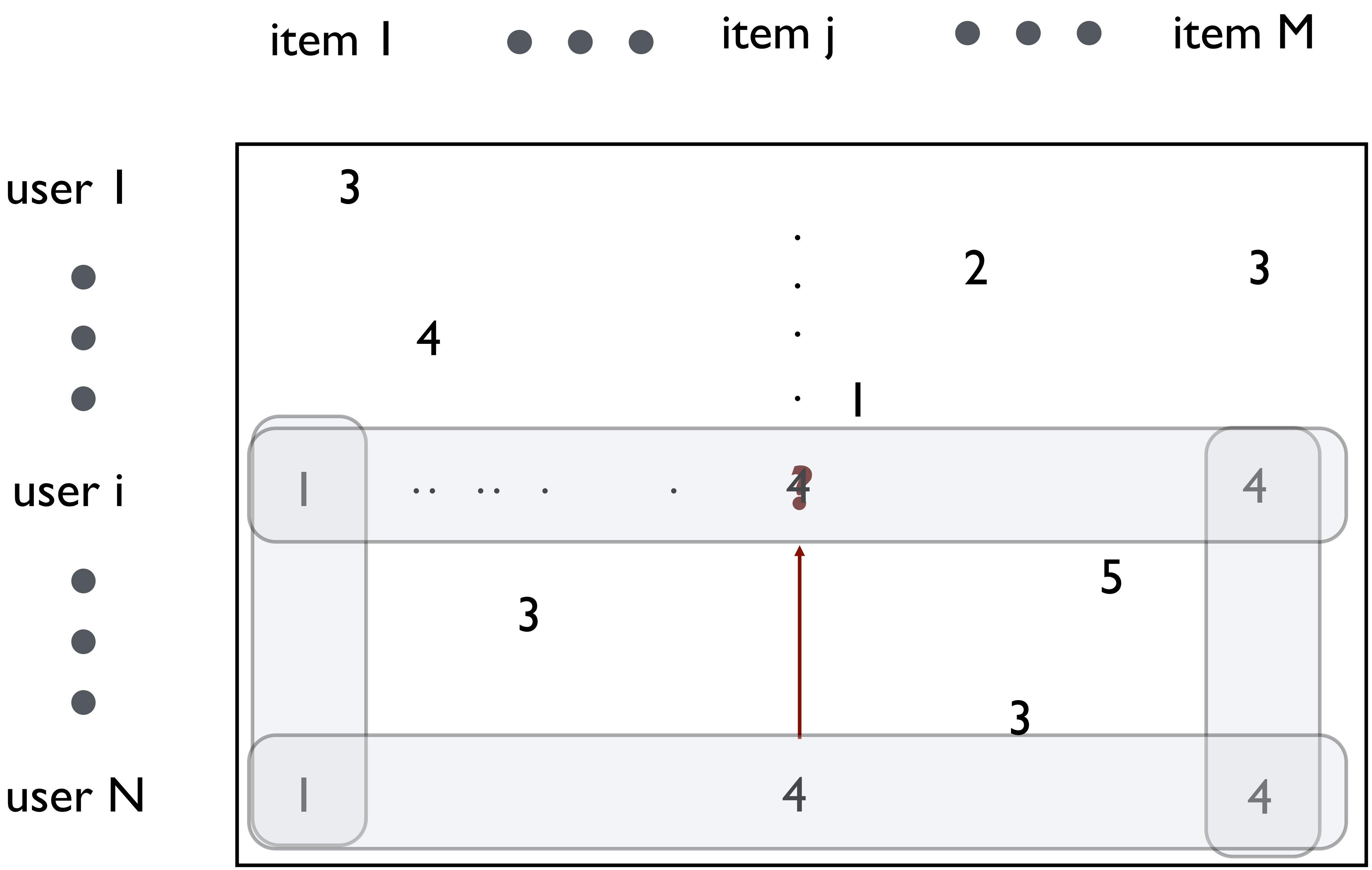
for each user (or item), find other users that are very similar to it

declare them as it's *personal* cluster

use the average over such a cluster to produce the estimate

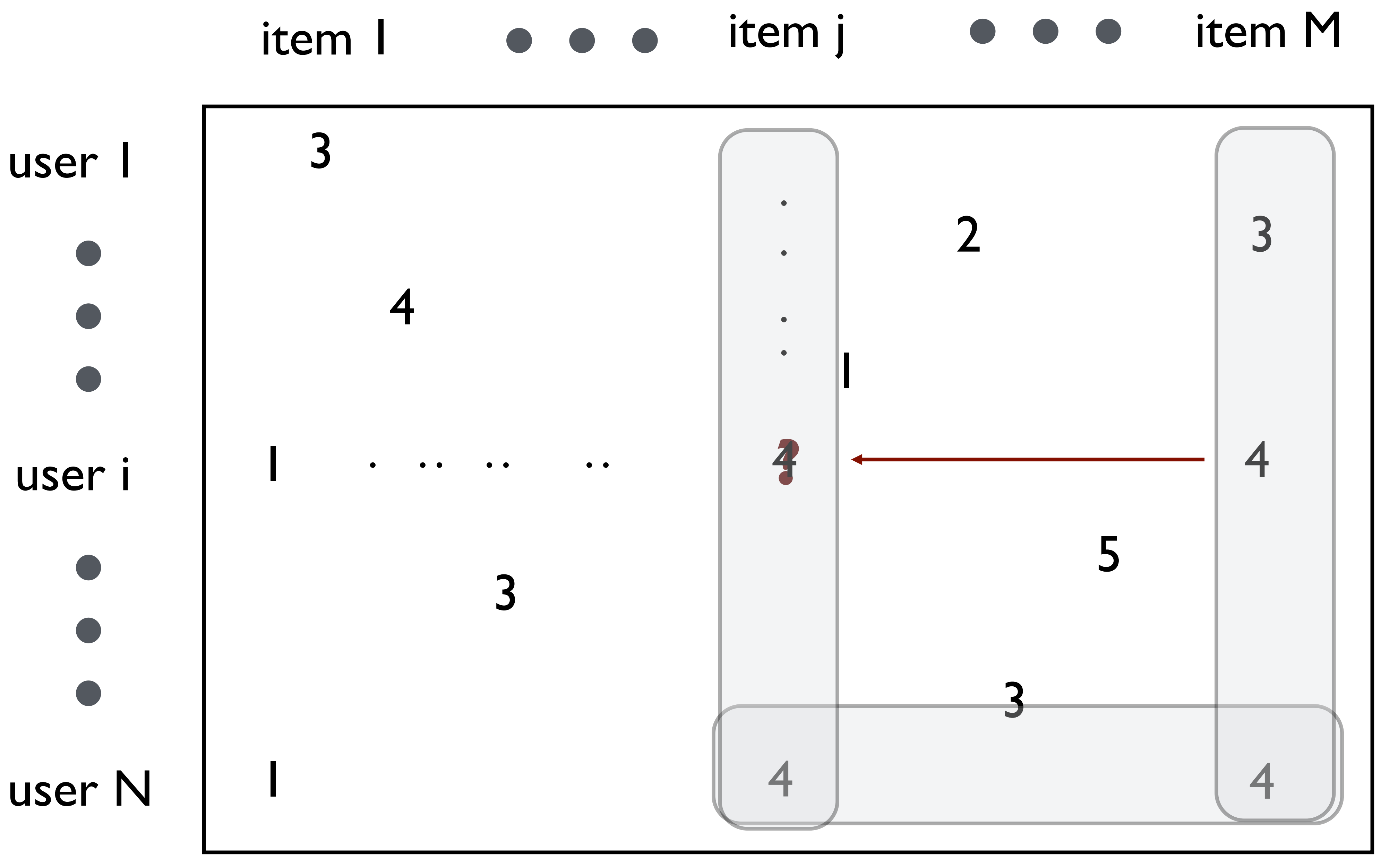
This is precisely what collaborative filtering attempts to do

Collaborative filtering



user-user collaborative filtering

Collaborative filtering



item-item collaborative filtering

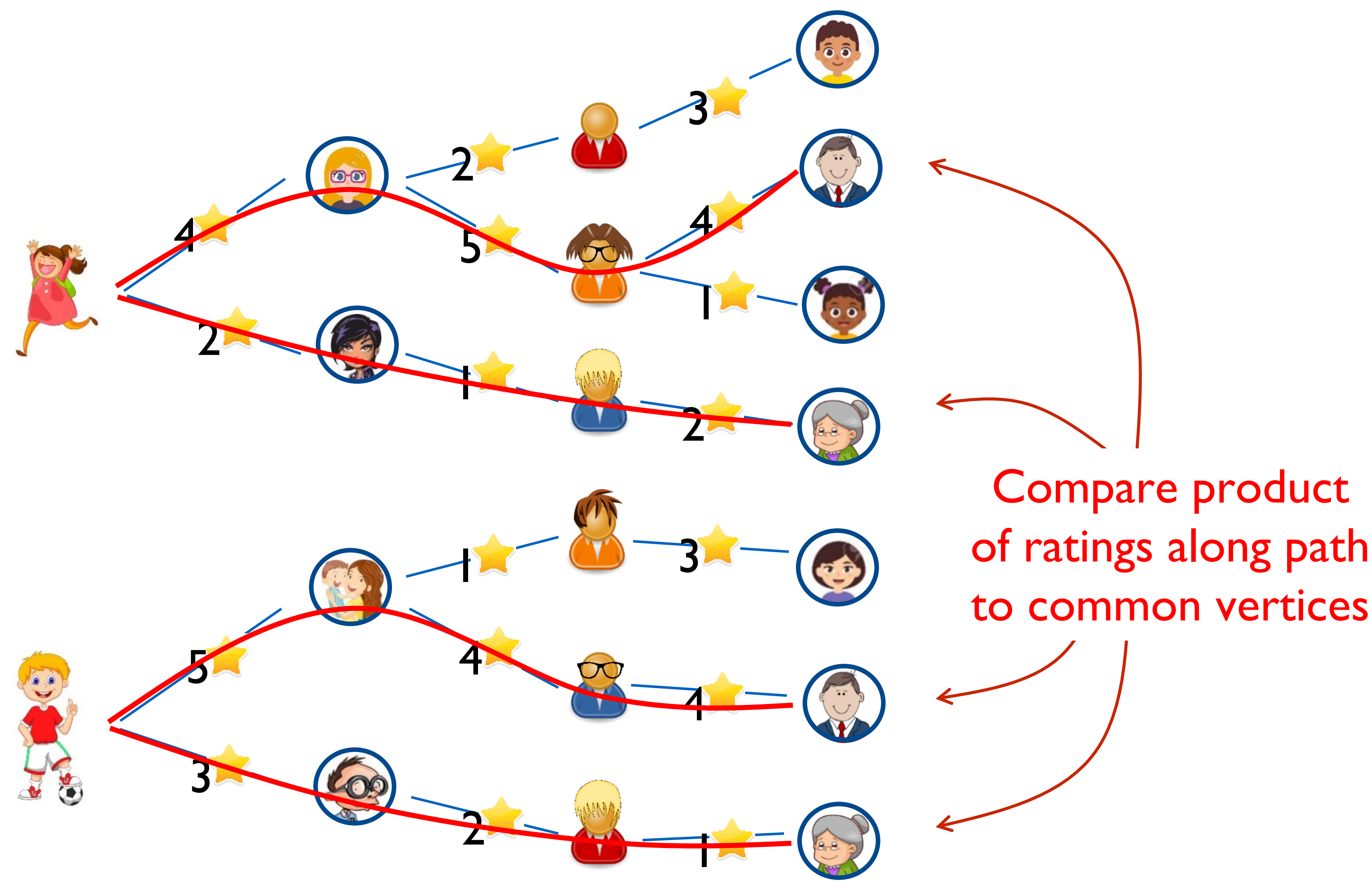
Collaborative filtering

Computing similarity requires common observations

(Birthday Paradox!)

What if observations are too sparse?

Iterative collaborative filtering



Thy Friend is My Friend!

Iterative collaborative filtering: In Summary

What if there are very few *neighbors* ?

Or they have not yet experienced the item of interest?

To overcome such sparse data challenge

Find users *similar* to user of interest

Next find users *similar* to these users

And continue *iterating* this procedure to find more *similar* users

till *enough similar* users are found

Use the experiences of such users to obtain the estimate

Collaborative filtering

Collaborative filtering and its variations

- Extensively used in practice
- Scalable implementation using "approximate nearest neighbors"
- Closely related to non-parametric nearest neighbor method
- Incremental and hence robust
- Interpretable:
  - You are being recommended *GoodFellas* because you liked *Godfather*
    - And, those who liked *Godfather also liked GoodFellas*

Explore: MovieLens Data

Exercise:

Apply collaborative filtering algorithm for MovieLens Data

Compare the estimation error of

- User-user collaborative filtering
- Item-item collaborative filtering
- Iterative collaborative filtering

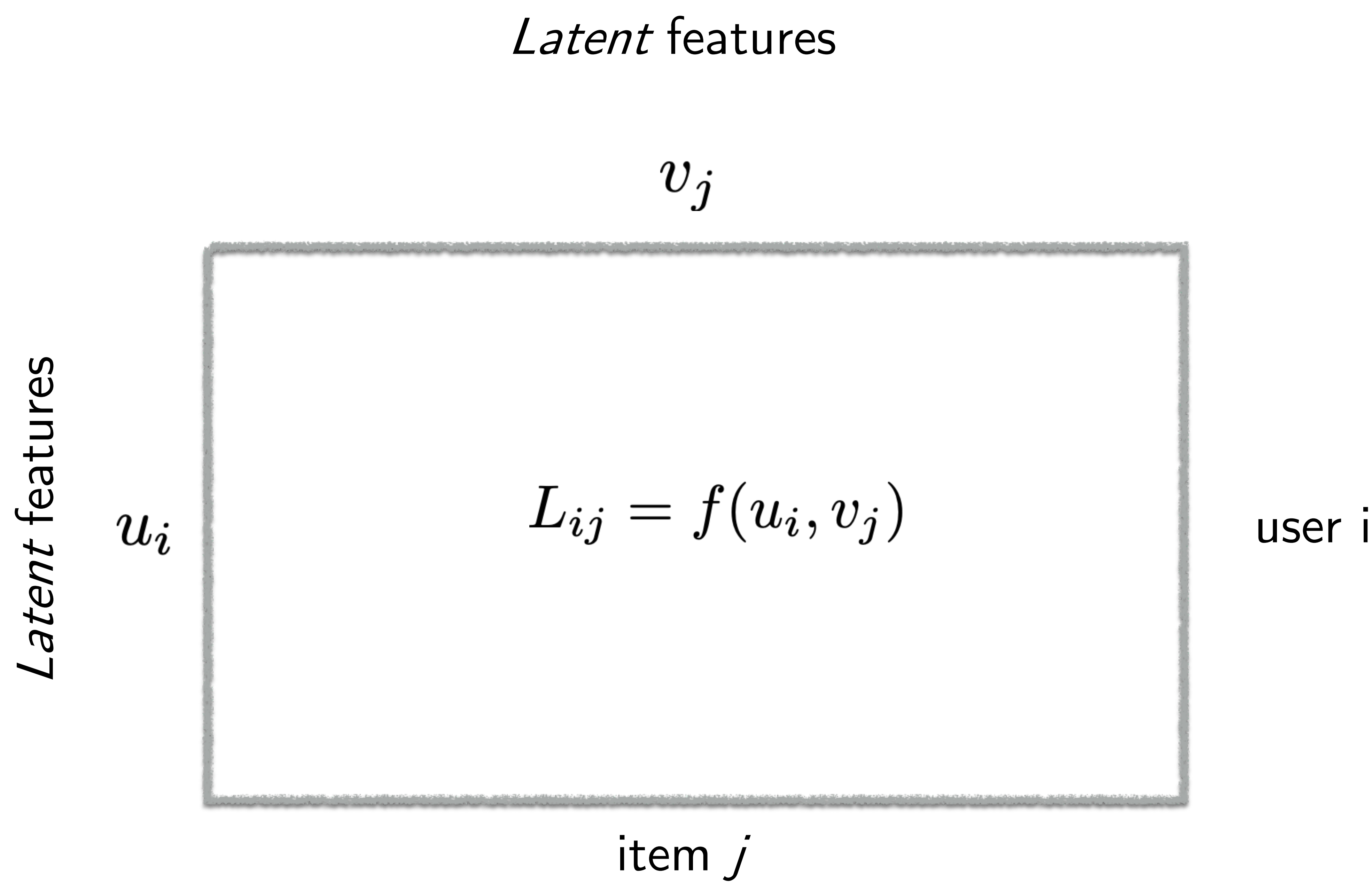
Data URL:

https://grouplens.org/datasets/movielens/100k

Module 3: Singular Value Thresholding

Matrix Estimation: Generic Model

Prediction problem: complete the matrix



Collaborative filtering is solving such a problem

Using effectively "nearest neighbors" approach!

Hey, L is a Matrix! Let's do Singular Value Decomposition

To estimate L_{ij} for any user *i* and item *j*

We need to *fill* missing values in a matrix

Now, any matrix obeys singular value decomposition: rank(X)=r

$$\underset{m \times n}{\overset{X}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}}} = \underset{m \times r}{\overset{U}{\begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix}}} \underset{r \times r}{\overset{S}{\begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix}}} \underset{r \times n}{\overset{V^{\mathsf{T}}}{\begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}}}$$

An example:

$$\begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} -2 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -2 & -1 \end{bmatrix}$$

Singular Value Thresholding

How to use SVD to estimate L_{ij} ?

Let $L = U\Sigma V^T$ (assuming we know it)

That is,

$$L_{ij} = \sum_{k \leq r} s_k u_{ik} v_{jk}$$

Therefore, if we know U, S and V

Then we can estimate all missing values

And de-noise observed values

Question: how do we find SVD of L?

Singular Value Thresholding

A natural algorithm: singular value thresholding

Step 1. Fill missing values in Y by 0

[better way to fill missing values?]

Step 2. Compute SVD

$$Y_{ij} = \sum_{k=1}^{\min(M,N)} s_k u_{ik} v_{jk}, \quad \text{for all } i, j$$

Step 3. Truncate SVD by keeping on top r components (+ normalize)

$$\widehat{L}_{ij} = \frac{1}{\hat{p}} \sum_{k=1}^{r} s_k u_{ik} v_{jk}, \quad \text{for all } i, j$$

where $\hat{p}$ is fraction of observed entries

[how to choose r?]

Explore: MovieLens Data

Exercise:

- Apply singular value thresholding algorithm for MovieLens Data
- Compare the estimation error
  - Across different thresholds
- How close is it to collaborative filtering?

Data URL:

https://grouplens.org/datasets/movielens/100k

Optimization perspective

Singular value decomposition: optimization perspective

Let U, V be solution of

$$\text{minimize} \sum_{i,j} (X_{ij} - \sum_{k=1}^{r} u_{ik} v_{jk})^2$$

$$\text{over} \quad u_{ik} \in \mathbb{R},\ 1 \le i \le N,\ 1 \le k \le r$$

$$\text{over} \quad v_{jk} \in \mathbb{R}, 1 \le j \le M,\ 1 \le k \le r.$$

Then, U and V are (effectively) left/right singular vectors

And. UV^T provides the best rank *r* approximate of X

Optimization perspective

This suggests the following algorithm

Find solution U, V of optimization problem

$$\text{minimize} \quad \sum_{(i,j):\text{obs}} (Y_{ij} - \sum_{k=1}^{r} u_{ik} v_{jk})^2$$

$$\text{over} \quad u_{ik} \in \mathbb{R},\ 1 \le i \le N,\ 1 \le k \le r$$

$$\text{over} \quad v_{jk} \in \mathbb{R}, 1 \le j \le M,\ 1 \le k \le r.$$

For any i, j, produce estimate $\widehat{L}_{ij} = \sum_{k=1}^{r} u_{ik} v_{jk}$

Optimization perspective

The algorithm uses *only* observed entries

and does not require filling missing values as in for SVD

The optimization problem can be solved

via iteratively solving for U and V

also known as *Alternating Least Squares*

Food for thought:

Will it converge?

Optimization perspective

Exercise: Singular Value Thresholding meets Alternative Least Squares

Initialize by filling missing values with 0

Singular Value Thresholding to obtain an estimate.

Use outcome to fill missing values and then perform Alt. Least. Sqs.

Iterate.

What are the advantages?

Use MovieLens and/or Yelp data to answer.

Matrix Estimation: Generic Model

Prediction problem: complete the matrix

Latent features

v_j

Latent features u_i

$$L_{ij} = f(u_i, v_j)$$

$$\text{low-rank:} \quad L_{ij} = u_i^T v_j$$

user i

item *j*

Problem reduces to learning "factorization" of the matrix

either through similarities or algebraic approaches

Appendix: Matrix Estimation with Neural Networks

Singular Value Thresholding

bi-linear function of latent features

"Generalized" Singular Value Thresholding

generic "activation" function of latent features

multiple layers

this provides neural network implementation

Exercise: compare performance with other methods