MATRIX-FACTORIZATION
RECOMMENDER-SYSTEMS
BI-CLUSTERING

RECOMMENDER SYSTEMS

- Many commercial/industrial applications
- Given a user and its past interaction with items, recommend them some new items
  - Movies, Music, Book, Video Games, etc.
  - Products on Amazon or any shop with past information
  - Posts/contents on Twitter, Facebook, Youtube, news media
  - …

RECOMMENDER SYSTEMS

- Intuition: How would you proceed to make recommendations?
  - e.g., Product to users
  - You have product descriptions, user descriptions, past user-product interactions
- What about a new user? A new product?
  - "Cold start" problem

CONTENT-BASED

- Classic approach: Content-based recommendation
  - We describe all our items using features
    - Movies genre, length, age rate, topics…
    - Object categories, price range, etc.
  - We recommend to users items having similar features to the ones they like
    - For instance, using supervised machine learning (classification or score regression)

- Often disappointing in practice
  - Finding useful descriptors is usually very hard
    - What makes you like/dislike a music/movie is more than a list of keywords
    - Somewhat arbitrary (is movie M a comedy? Book B a child book? 2 people might disagree)
  - Very costly on large catalogs
    - Impossible for social media, but also Amazon, YouTube..

COLLABORATIVE FILTERING

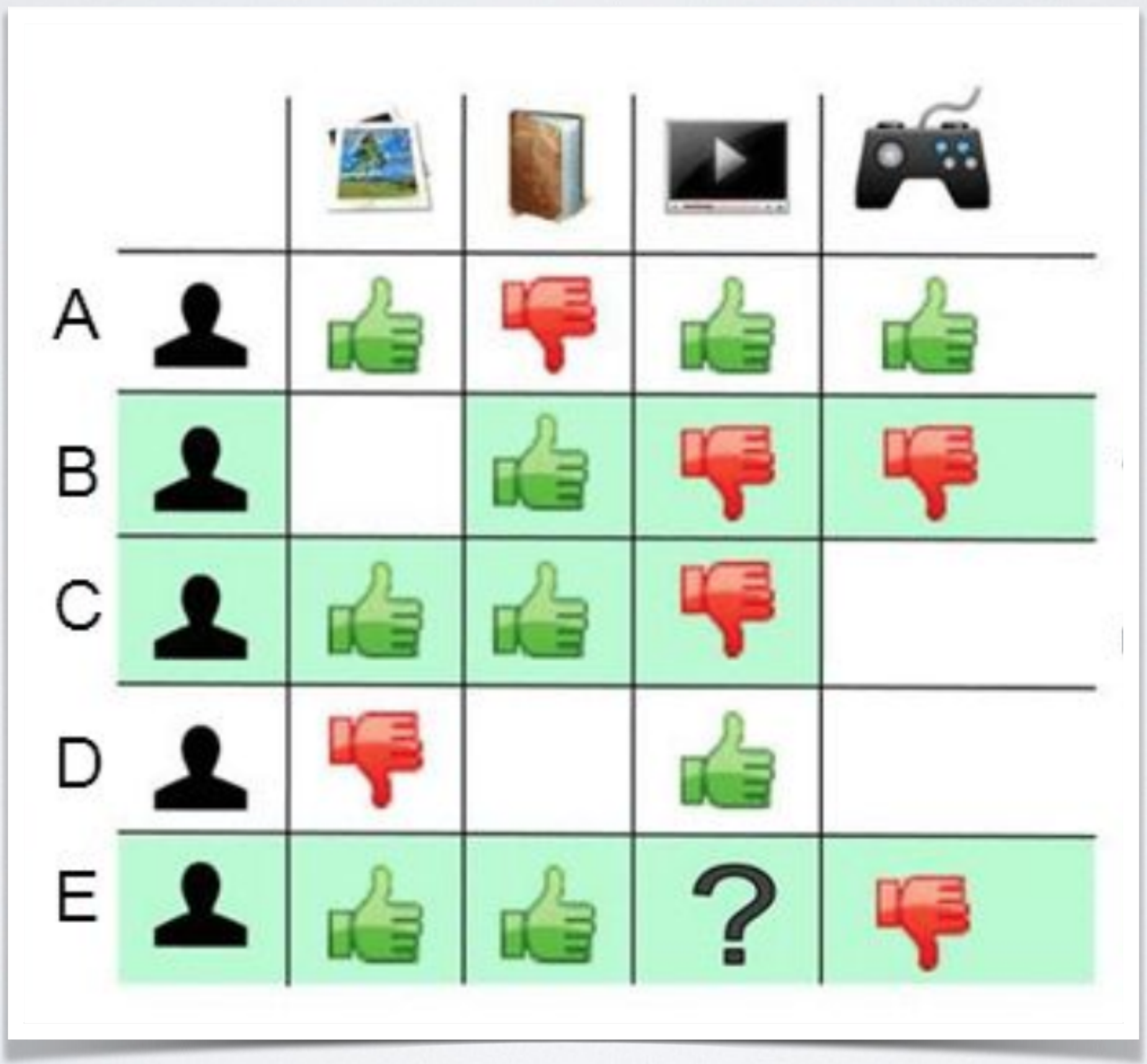
- Solution: **Collaborative filtering**

- Principle:
  - To evaluate if two items are similar, instead of comparing manually chosen descriptors (genre, etc.), we compare the users who have interacted with them
  - =>Users themselves become the features

- The definition of similarity **emerges** from the **collaborative** efforts of all users

- *Tell me what you like, I'll tell you who you are*

COLLABORATIVE FILTERING



DATA

- We model observed data as a matrix of size $U \times I$
  - U users
  - I items
- $X(u, i)$=user/item interaction
  - Buy, watch, clic, like, vote, etc.
- Users could be treated as any feature, but they have some specificities
  - Values are sparse:
    - Missing values in all rows and columns (no user rates all items, no item is rated by every user)
  - Both Users or Items can be used as variables or observations (rows/columns)

DATA COMPLEXITY

- Data form:
  - Binary vote
    - 1 and 0 are both reliable (rare)
  - Like, Heart, Watched, Bought, Listened, etc.
    - 1 is reliable information, but 0 and nan are not differentiable.
  - Note (e.g., 1 to 5 stars, etc.)
    - Often imbalanced between 4/5 (frequent), 1/2 (less frequent)
    - Missing values and 0 are correlated (people rate what they watch, and watch what they like)

DATA COMPLEXITY

- Users can have different labeling standards
  - "Good" for one might correspond to "excellent" for another
    - Some users put a like/share everything they find above-average
    - Other users will only like/share what they find exceptional
    - Same for scores: some users never give maximal notes, while others use only the maximal note

- Normalizing by users?
  - We don't care if the score is good, we consider if it is higher or lower compared with other scores from the same user

- Normalizing by item?
  - We don't care anymore if the score is good, we want to know if it is better than for other users

USER/ITEM BIAS TERM

Normalizing both aspect together

BIAS TERM

- We estimate the baseline score for (u, i) from values b_u and b_i
    - b_u captures the tendency of u to give high or low marks
    - b_i captures the tendency of i to have low or high marks
    - $r(u, i)$: rate given by u to i
    - Minimize reconstructing error
  - $$\sum_{r_{ui}} \left(r_{ui} - (\mu + b_u + b_i)\right)^2$$
    - μ: average note (all users, all items)
  - b cannot capture how much a particular user likes a particular movie.
    - Captures only tendencies of users/ of items

- Solved by gradient descent

BIAS TERM

- In practice, add regularization terms
    - $$\sum_{r_{ui}} \left(r_{ui} - (\mu + b_u + b_i)\right)^2 + \lambda \left(b_u^2 + b_i^2\right).$$
    - Regularization tends to impose low b.

USER-BASED KNN

USER-BASED KNN

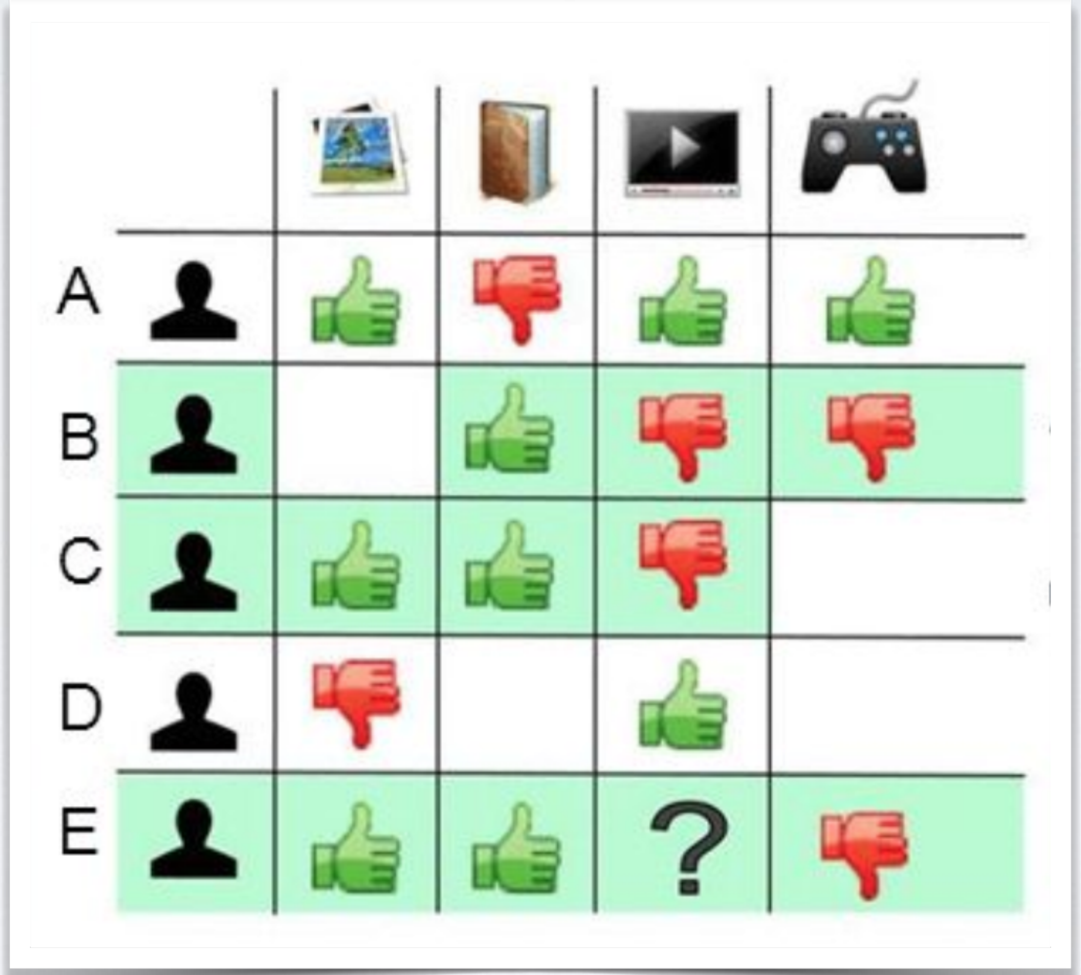
- KNN: K-Nearest-Neighbors
  - Simple yet powerful method popular in classification task
    - 1)Find k most similar items (neighbors) to item i.
    - 2)Each neighbor "vote" for its target => average/mode of targets of neighbors

- Application to **user-based** collaborative filtering
  - 1) Find k most similar users (neighbors)
  - 2) Each neighbor "vote" for the products they liked
    - Average notes
    - Count of 1 for binary data (like, etc.)
  - Usually, votes weighted by similarity to the original user

USER-BASED KNN

Similarity to E

-1
2
2
-1



USER-BASED KNN

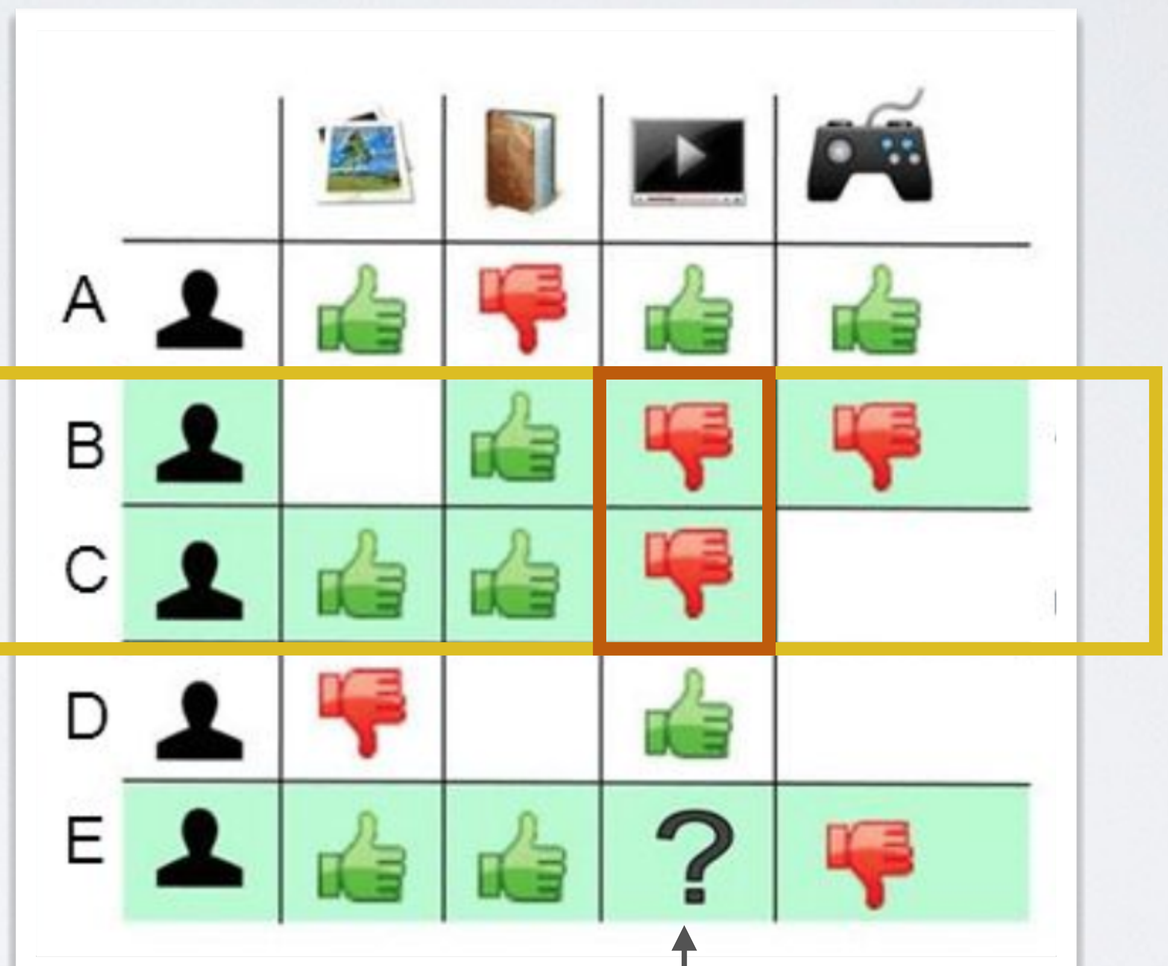
Similarity to E

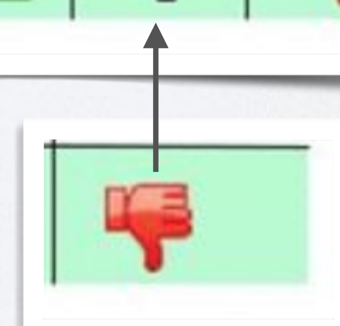
-1

2

2

-1





$(2 * -1)/2 = -1$

SIMILARITY

- How to compute the similarity between users?
  - Euclidean distance => No, because of sparsity (most values are 0)
    - Think of a user with few likes {0,1}. They are very distant from users having many like, since each difference adds distance.
  - Number of similar votes only? $= R_u \cdot R_v$
    - (R_u=>vector of all votes of u)
    - Now users with many likes are similar to everyone

- Solution:
  - (Binary & Notes) => Cosine Similarity $\dfrac{R_u \cdot R_v}{|R_u||R_u|}$
  - (Binary) Jaccard Similarity => $\dfrac{R_u \cdot R_v}{|R_u| + |R_u| - R_u \cdot R_v}$
  - (Notes) MSD=>Mean Squared Difference <u>when both notes present</u>

ITEM-BASED
COLLABORATIVE FILTERING

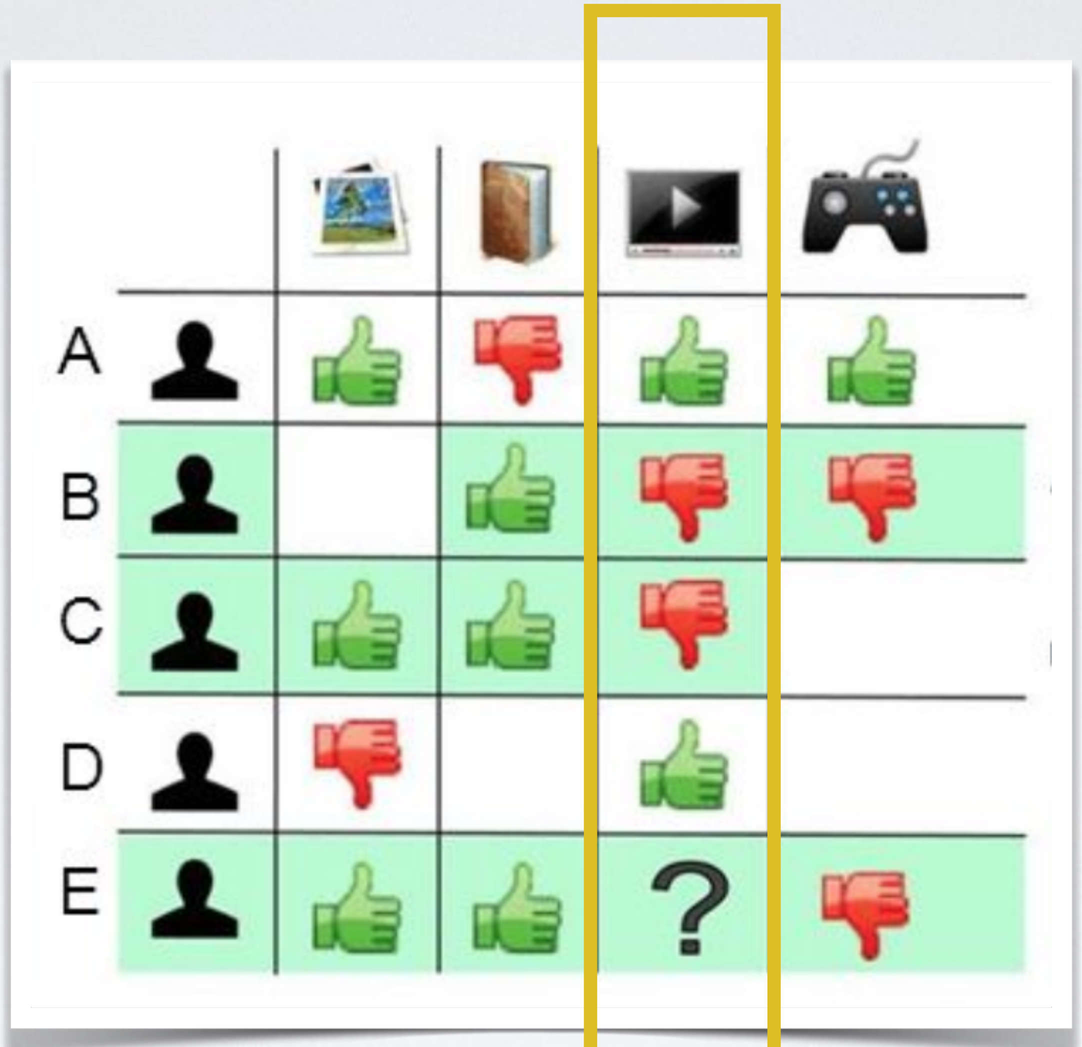
ITEM-BASED

- User-based collaborative filtering has weaknesses in practice
  - Scalability: Users change a lot =>Need to recompute KNN on the whole database very frequently
  - Users with little info will have neighbors with little info too
    - Imagine you liked movies M1 and M2. The 20 most similar users will like exactly M1 and M2, maybe 1 movie more.
    - =>We will learn based on few info
- => Move to Item-based Collaborative filtering
  - Compute similarity between **items**, based on votes
  - Then compute

ITEM-BASED

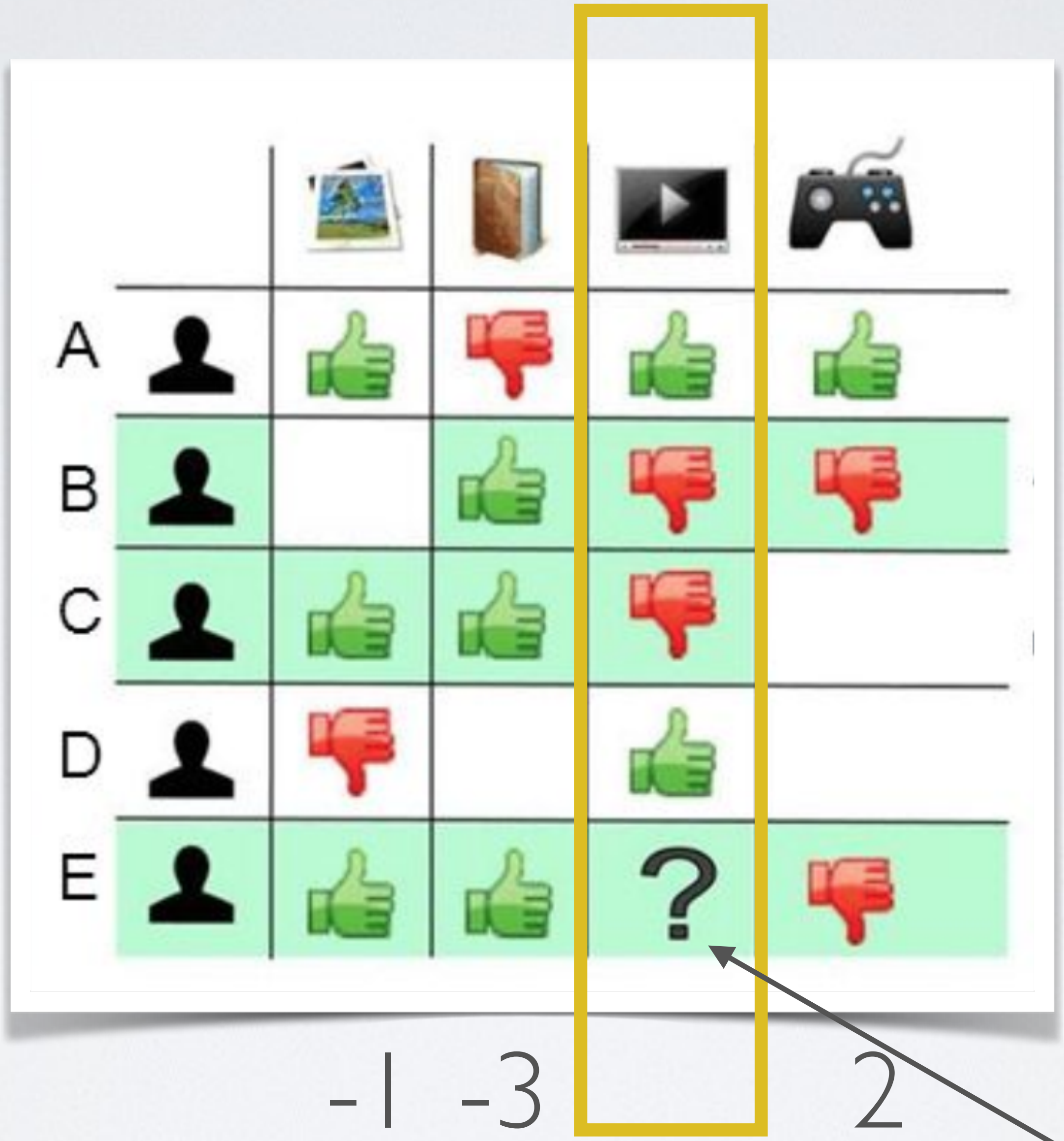
- 1)Compute similarity between **items**, based on votes
- 2) Then compute for each user, the most similar items
  - Based on the items they liked

ITEM-BASED



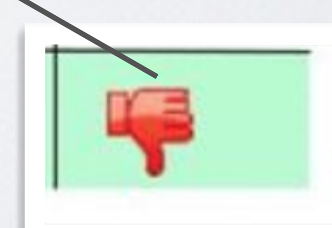
-1 -3 2

ITEM-BASED



-1 -3 2

$=(1*(-1)+1*(-3)-1*2)/3 \Rightarrow -2$



ITEM-BASED

- Original Amazon patented method introduced in 1998
- Strengths
  - Distances between items can be precomputed at fix interval, do not change too quickly
  - Distances between items robust, lot of information (appart from new items)

MATRIX FACTORIZATION COLLABORATIVE FILTERING

LATENT FACTORS

Matrix factorization in **dense** matrices
(i.e., mostly non-zero values)

LATENT FACTORS

- A popular problem in Data Mining
- Given two types of data
  - Locations and Dates (T°, mortality in cities along week/year…)
  - Terms and Documents (Topic-modelling)
  - …
- Unsupervised task
  - How to best reconstruct the data
  - By assigning a “latent variable” to each item

MATRIX FACTORIZATION

- **Matrix Factorization**
  - We start with an original matrix A, typically item/user matrix
  - We search for 2 matrices U,V, such as to minimize a cost function $L(A, UV)$
    - With UV a matrix multiplication
  - Or with the SVD technique, 3 matrices, $U\Sigma V$, with Σ giving the relative importance of factors.

- If the dimension of A is $X \times Y$, dimensions of
  - $U => X \times D$
  - $V => D \times Y$
    - With D a parameter, corresponding to a number of ***latent variables***/***embedding dimensions***

- Same principle as PCA dimensionality reduction

MATRIX FACTORIZATION

- Dimensions can be understood as *latent variables*, i.e., features representing some semantic notion

- For instance, in movies, latent variables could capture
  - Horror-ness, comedy-ness, adult-ness, etc.
  - Each user has a score in each of these features (enjoy horror=1, comedy=0.2)
  - Each movie too (is horror=1, is comedy=1.5)
  - =>(user, movie)=>combination of match in each category

NETFLIX PRIZE

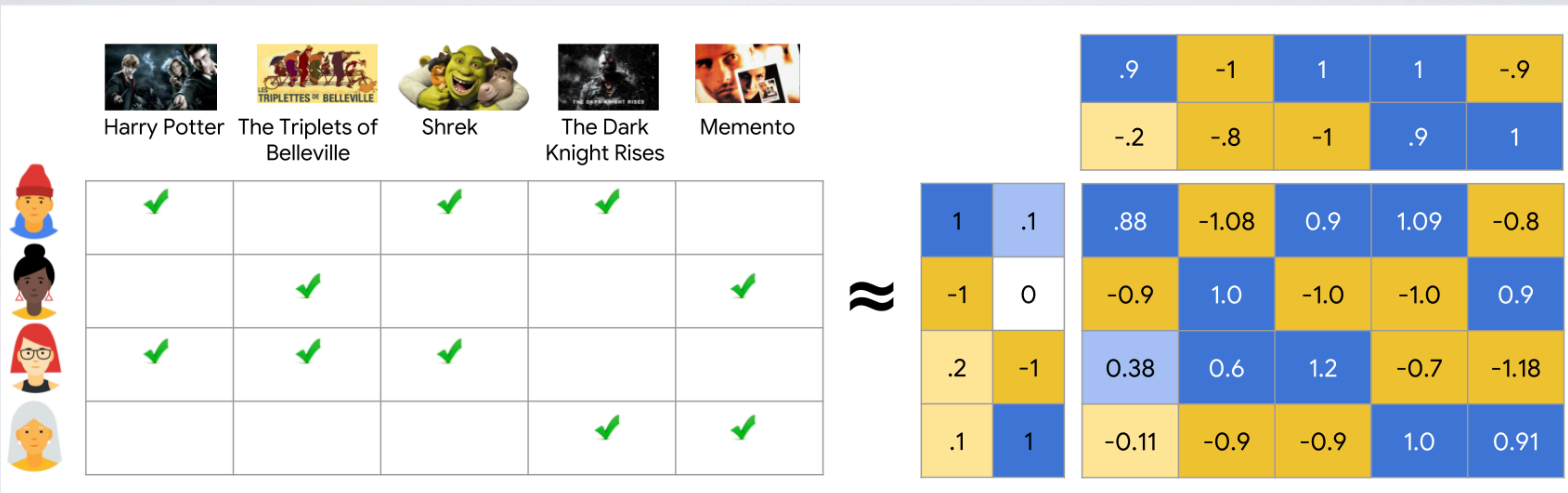
- Worldwide competition to improve Netflix recommendation
  - Cash prize, 1 Million $
  - 2006 to 2009 (Objective of reducing RMSE on scores by 10% compared with Netflix own method)

- Winning method: Stacking of multiple recommendation systems

- But the single most successful approach: Matrix decomposition
  - 2 matrices only, special treatment of sparse matrices

https://intoli.com/blog/pca-and-svd/

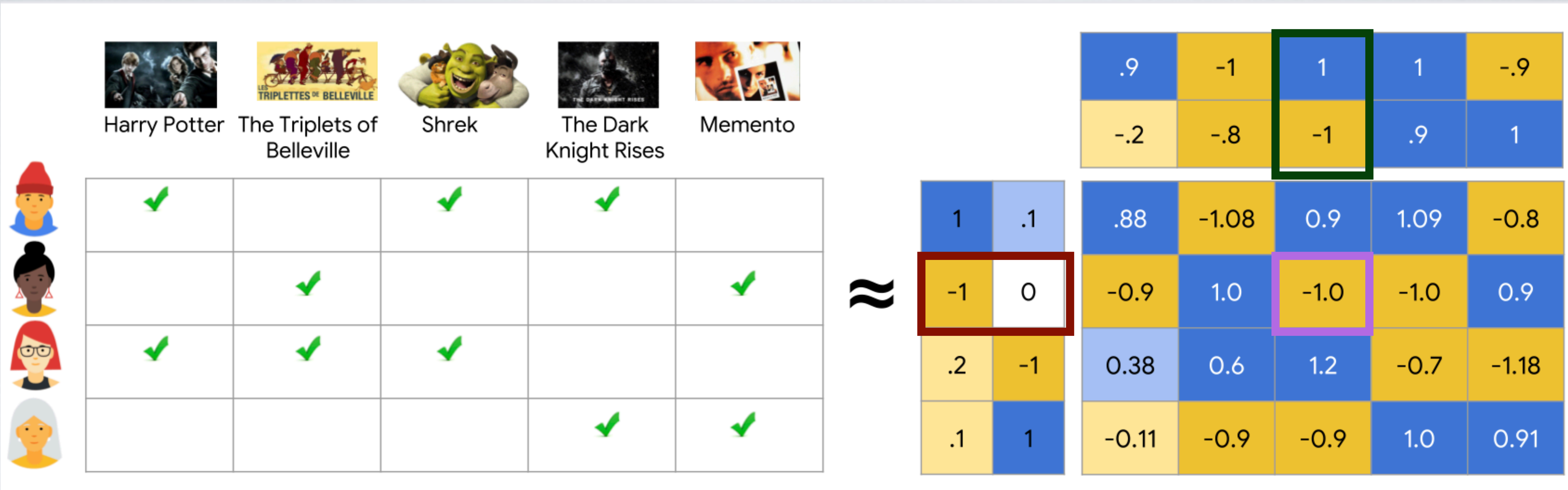
MATRIX FACTORIZATION



2 latent variables

https://developers.google.com/machine-learning/recommendation/collaborative/matrix

MATRIX FACTORIZATION



Vector representing user 2, u2

Vector representing item 3, i3

Multiply the two vectors to reconstruct estimated value(u2,i3)

https://developers.google.com/machine-learning/recommendation/collaborative/matrix

OBJECTIVE FUNCTION

- The *classic* SVD would correspond to using as a loss the mean-squared error
  - **Having 0 where we have no data** (like/rating)

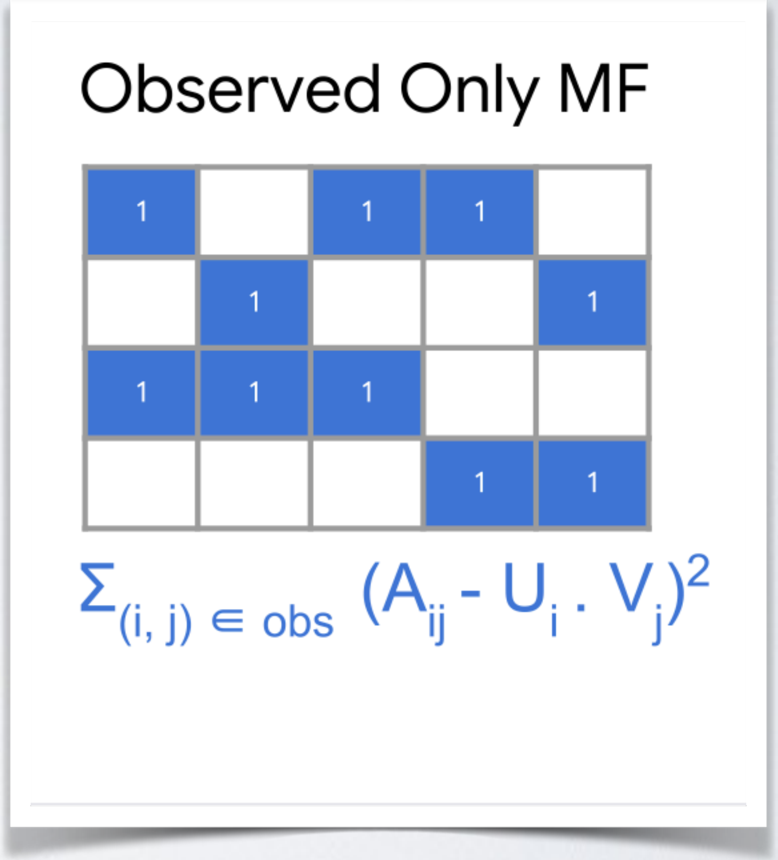
SVD

1	0	1	1	0
0	1	0	0	1
1	1	1	0	0
0	0	0	1	1

$$|A - U V^T|_F^2$$
$$= \Sigma_{(i,\, j)} (A_{ij} - U_i \,.\, V_j)^2$$

OBJECTIVE FUNCTION

- The recommendation based Matrix Factorization has an adapted loss,
  - **Computed only on non-zero values**
  - Solve sparsity, i.e., missing values



OBJECTIVE FUNCTION

Observed Only MF

1		1	1	
	1			1
1	1	1		
			1	1

$$\Sigma_{(i,\, j) \in obs} (A_{ij} - U_i \,.\, V_j)^2$$

Weighted MF

1	0	1	1	0
0	1	0	0	1
1	1	1	0	0
0	0	0	1	1

$$\Sigma_{(i,\, j) \in obs} (A_{ij} - U_i \,.\, V_j)^2 + w_0\, \Sigma_{(i,\, j) \notin obs} (0 - U_i \,.\, V_j)^2$$

SVD

1	0	1	1	0
0	1	0	0	1
1	1	1	0	0
0	0	0	1	1

$$|A - U V^T|_F^{\,2} = \Sigma_{(i,\, j)} (A_{ij} - U_i \,.\, V_j)^2$$

A variant has a parameter to combine both (Weighted Matrix Factorization)

https://developers.google.com/machine-learning/recommendation/collaborative/matrix

OPTIMIZATION

- To find the two matrices, we use a greedy approach
  - Typically the Weighted Alternating Least Square (WALS)
    - 1)Initialize values at random
    - 2)Fix $\boldsymbol{U}$ and solve for $\boldsymbol{V}$
    - 3)Fix $\boldsymbol{V}$ and solve for $\boldsymbol{U}$
    - Repeat 2 and 3 until convergence
  - Solving in 2 and 3 is equivalent to doing linear regression for each component

OPTIMIZATION

$$\begin{array}{rc} & \begin{array}{ccc}\text{Product 1} & \text{Product 2} & \text{Product 3}\end{array} \\ \begin{array}{r}\text{user 1}\\ \text{user 2}\end{array} & \begin{bmatrix}0.5 & ? & 4\\ 1 & 3 & 5\end{bmatrix}\end{array} = \begin{bmatrix}u_1\\ u_2\end{bmatrix}\begin{bmatrix}p_1 & p_2 & p_3\end{bmatrix}$$

$$\begin{array}{rc} & \begin{array}{ccc}\text{Product 1} & \text{Product 2} & \text{Product 3}\end{array} \\ \begin{array}{r}\text{user 1}\\ \text{user 2}\end{array} & \begin{bmatrix}0.5 & ? & 4\\ 1 & 3 & 5\end{bmatrix}\end{array} = \begin{bmatrix}1\\ 1\end{bmatrix}\begin{bmatrix}p_1 & p_2 & p_3\end{bmatrix}$$

Arbitrary initialization

$$p_1^* = \operatorname{argmin}\,(0.5 - p_1)^2 + (1 - p_1)^2 \tag{6}$$

$$p_2^* = 3$$

$$p_3^* = \operatorname{argmin}\,(4 - p_3)^2 + (5 - p_3)^2 \tag{7}$$

$$P = \begin{bmatrix}0.75 & 3 & 4.5\end{bmatrix}$$

$$U = \begin{bmatrix}0.7461\\ 1.7966\end{bmatrix} \qquad P = \begin{bmatrix}0.758 & 2.5431 & 4.7999\end{bmatrix}$$

MF + REGULARIZATION

- As with many machine learning tasks, we can introduce regularization to avoid overfitting
  - Due to the large number of parameters, regularization is important
- The objective to solve becomes:
  - $$\sum_{r_{ui} \in obs} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(||q_i||^2 + ||p_u||^2 \right)$$
    - q_i, p_u are latent vectors, $\hat{r}_{ui} = q_i p_u^T$
  - λ controls the strength of the regularization
    - Tries to minimize information in the vectors, avoid overfit

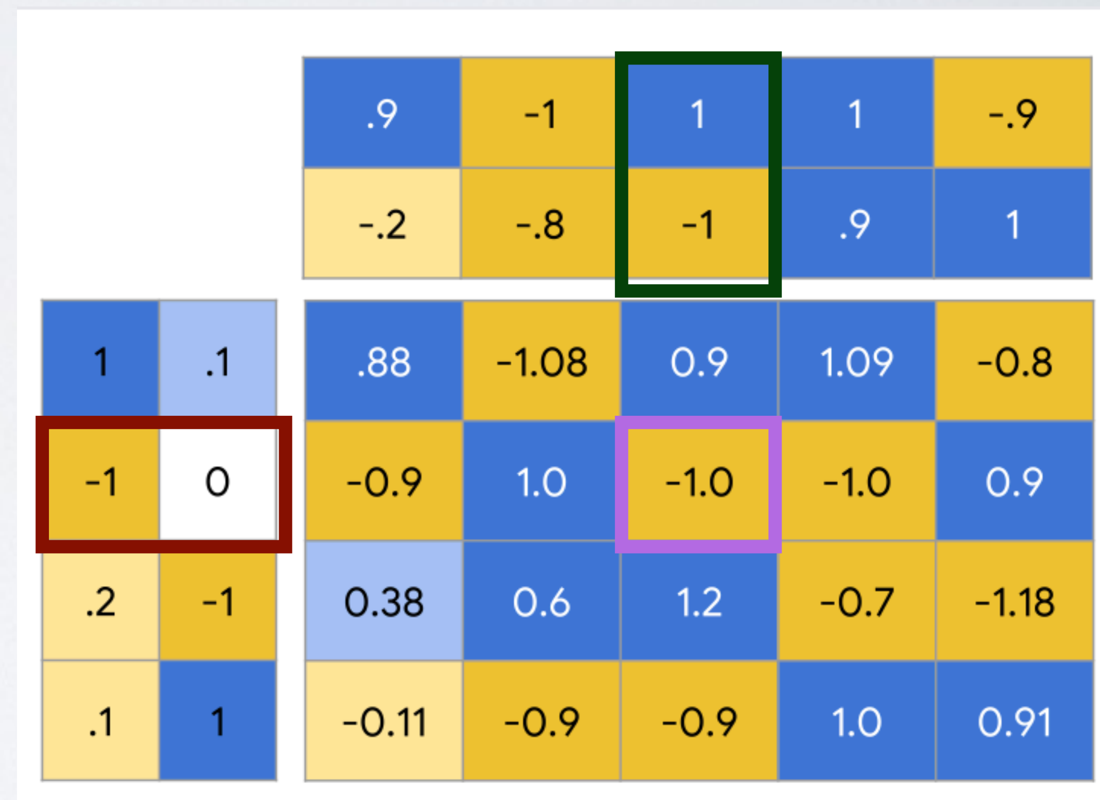
MF + BASELINE

- As mentioned before, it is also important to take into account the variability of users and of items
  - We want to predict what cannot be simply predicted by
    - Movies being good/bad
    - Each actor tendency to give good/bad scores

- The objective to solve becomes:
  - $$\sum_{r_{ui} \in obs} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 \right)$$
  - b_i and b_u = user baselines
  - $\hat{r}_{ui} = q_i p_u^T + \mu + b_i + b_u$

MF RECOMMENDATION

- From the two partial matrices, we can compute any value by multiplying the corresponding vectors
- Recommending for a user consists in picking
  - In the user row
  - The highest computed values



NETFLIX PRIZE

- A few other elements were taken into account in the Netflix Prize winning strategy
  - Temporal aspects (how long since this product was rated…)
  - Sequential aspects
    - Watch episode1 then episode 2. Contrary unlikely.
- Fine parameter tuning, clever stacking…

EVALUATION OF RECOMMENDER SYSTEMS

EVALUATION

- Recommendation evaluation use similar quality scores as supervised machine learning evaluation
  - RMSE, Precision@k, AUC, etc.

EVALUATION

- In practice, two ways to evaluate, hiding users or hiding pairs(u,i)

- Hiding pairs (u,i)
  - Hide random (u,i) pairs, ensuring a minimal number of visible ratings per user and items
  - Evaluate the recommendation on those removed pairs.

- Hiding users
    - If possible, even keep the most recent users hidden: prediction at time *t*
  - 1)We train with full data on a fraction of users
  - 2)We validate with test users, considered "new"

-

OTHER RECOMMENDATION QUALITY CRITERIA

- Diversity of recommendation
  - e.g., maximize average cosine distance between 2 items recommended to a same user (among top-5)
- Coverage
  - e.g., fraction of all items recommended at least once…
- Personalization
  - e.g., maximize average cosine distance between recommendations made to different users

MF VARIANT: NMF

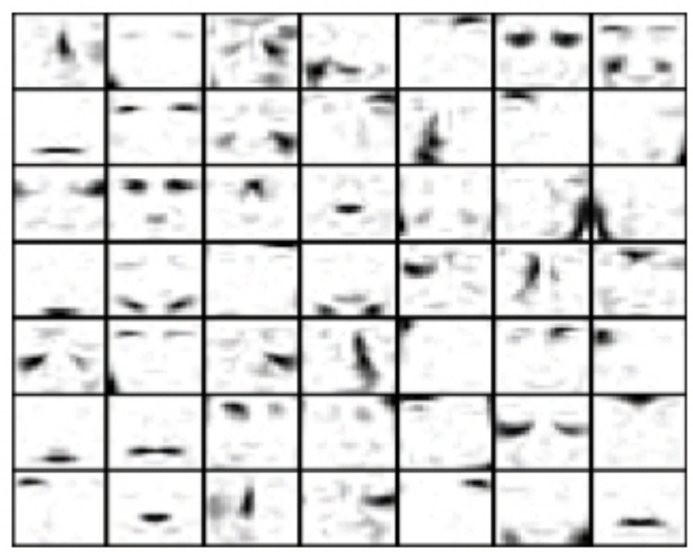
Non-negative Matrix Factorization

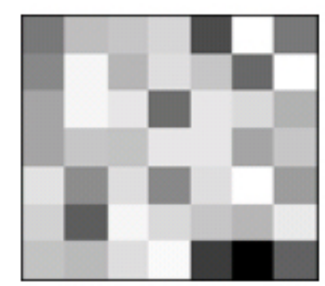
NMF

- A strength of Matrix Factorization is that it produces latent variables which, in theory, can be interpretable.
- A weakness of classic MF is that these variables can cancel each other, if one is positive and the other negative
- In NMF (Non-negative MF), we impose that all variables values must be positive. Of course, the Matrix to decompose must be positive too.
  - Imposes additive combinations

NMF

$$\underbrace{X(:,j)}_{j\text{th facial image}} \approx \sum_{k=1}^{r} \underbrace{W(:,k)}_{\text{facial features}} \; \underbrace{H(k,j)}_{\substack{\text{importance of features} \\ \text{in } j\text{th image}}} = \underbrace{WH(:,j)}_{\substack{\text{approximation} \\ \text{of } j\text{th image}}} .$$





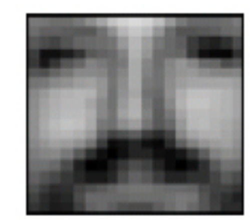


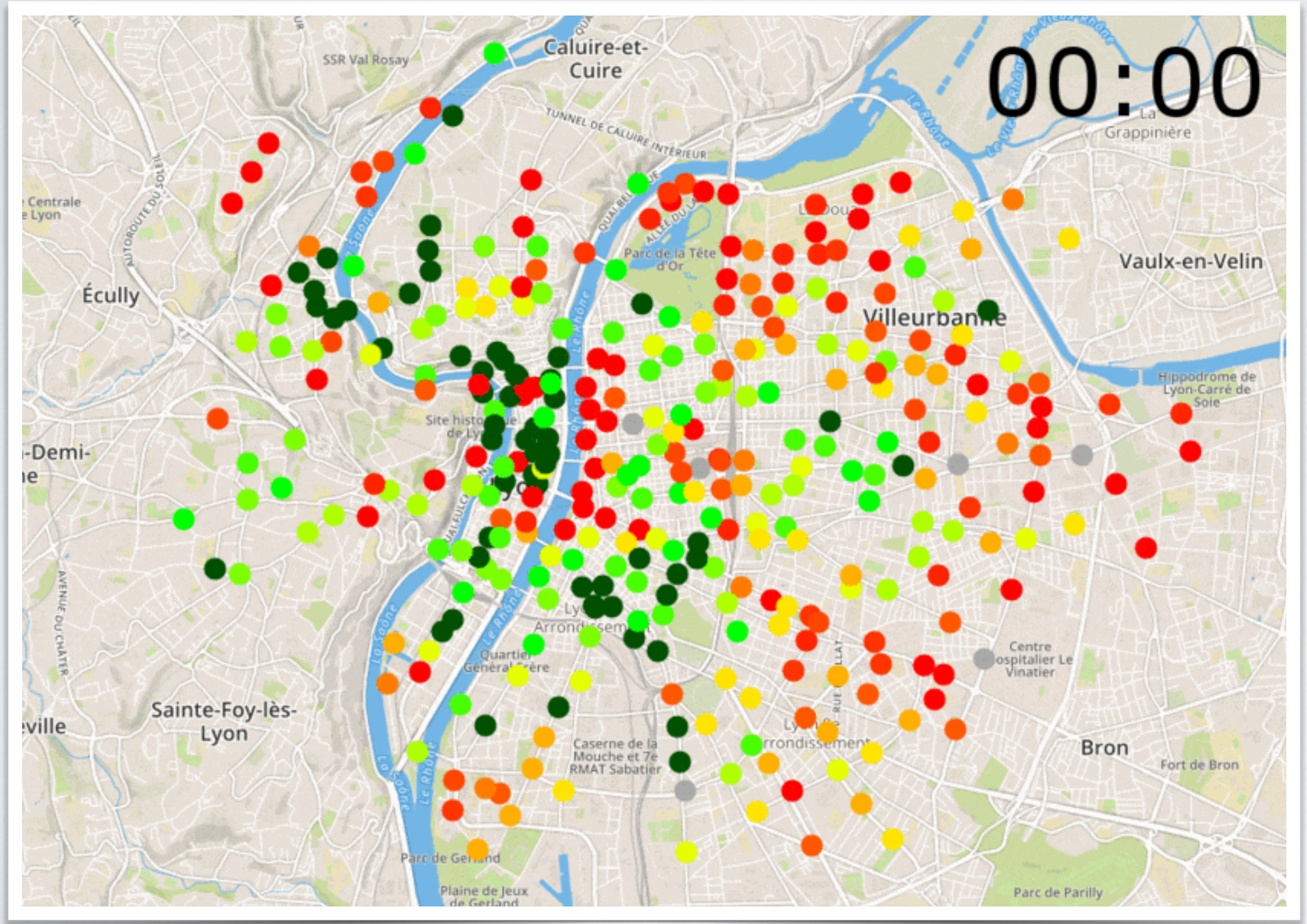
Figure 1: Decomposition of the CBCL face database, MIT Center For Biological and Computation Learning (2429 gray-level 19-by-19 pixels images) using $r = 49$ as in [79].

BICYCLE SHARING SYSTEMS

Docking stations

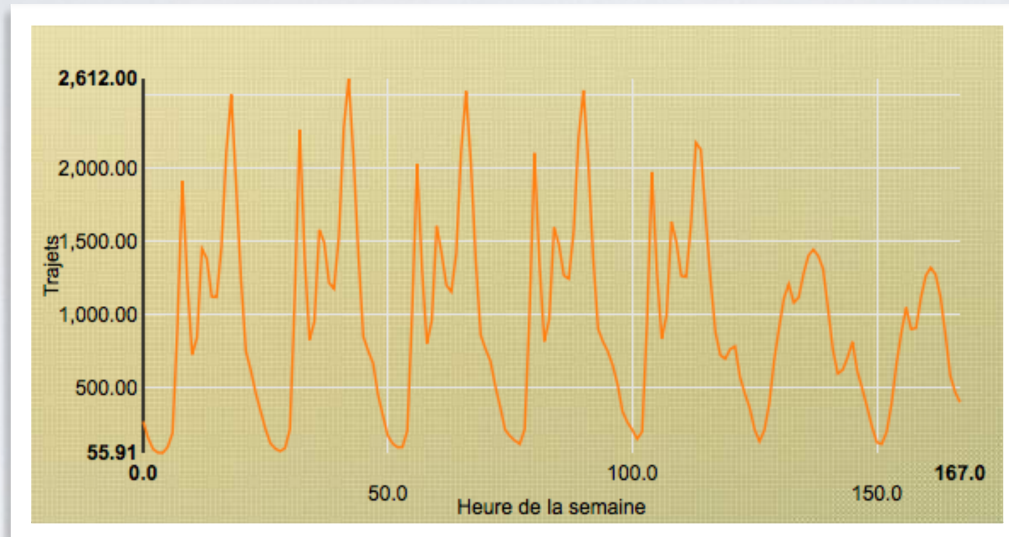
Bicycle trips

DATA

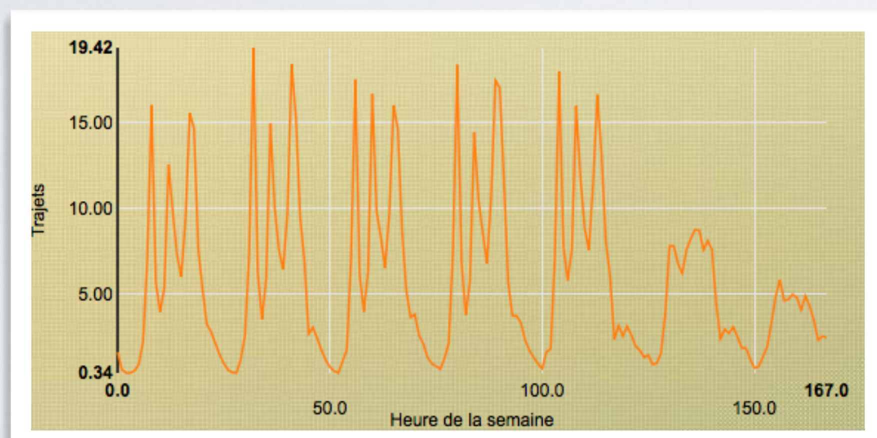


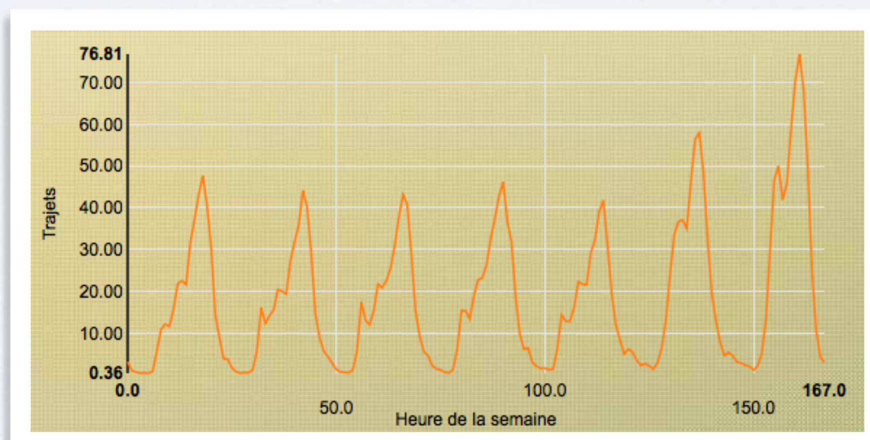
Red: empty

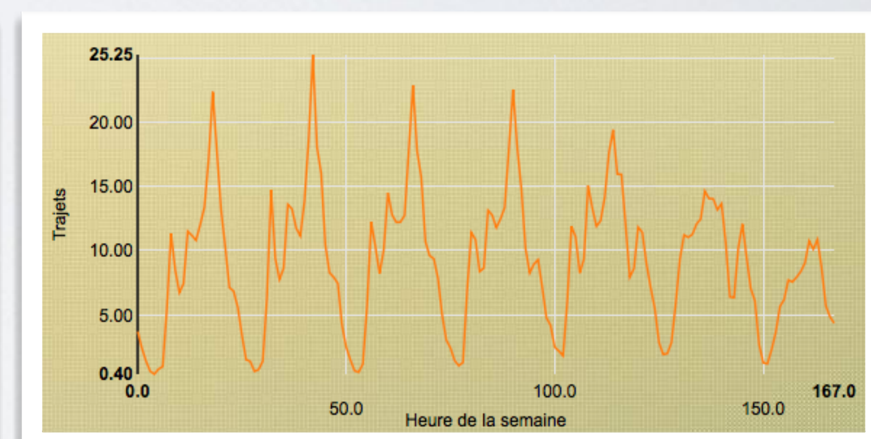
Green: full



Cumulated







Part Dieu

Tête d'or

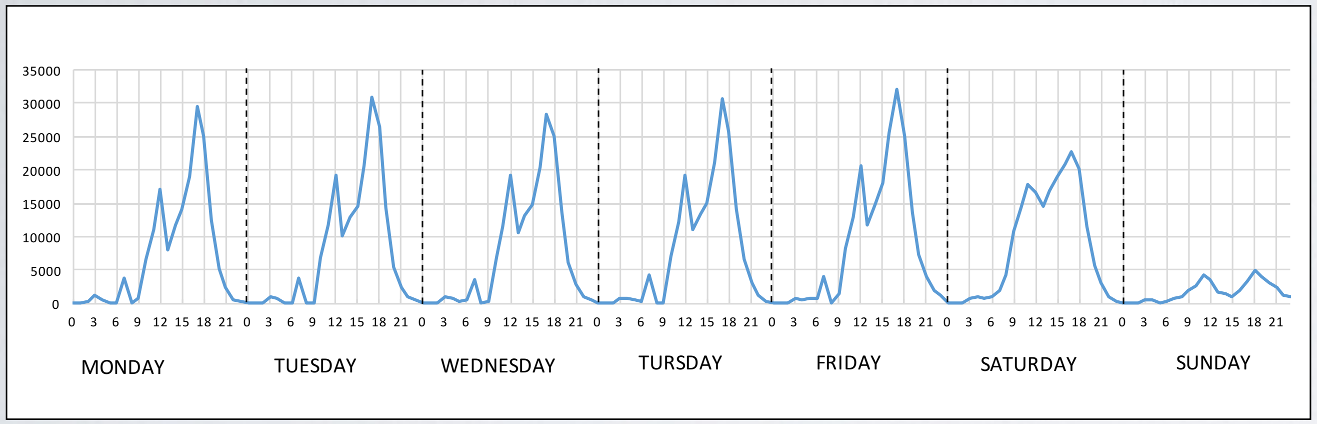
Guillotière

Hours of the typical week

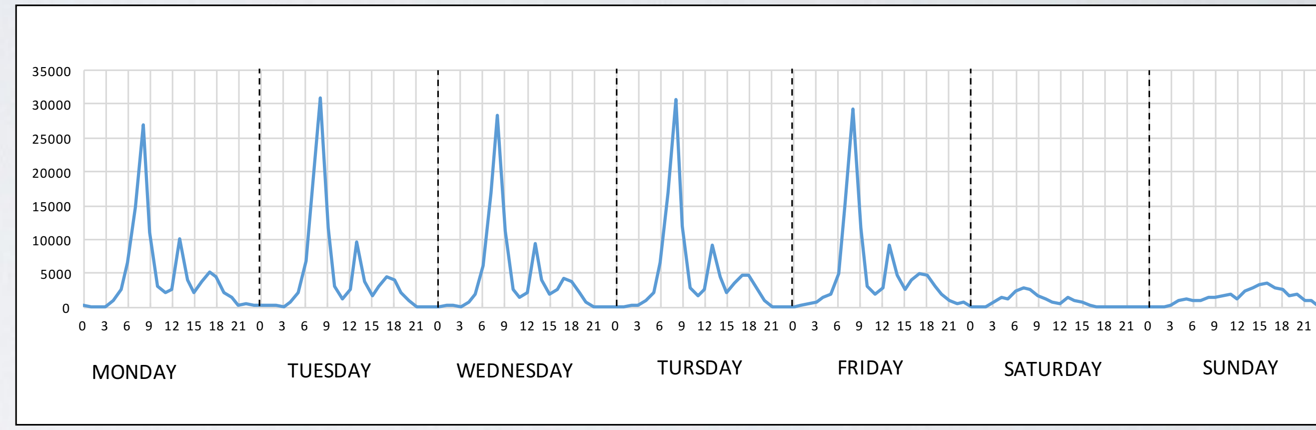
Entities (station)

	t1	t2	t3	t4	t5	t6	…		t168
e1									
e2									
e3									
e4									
…									

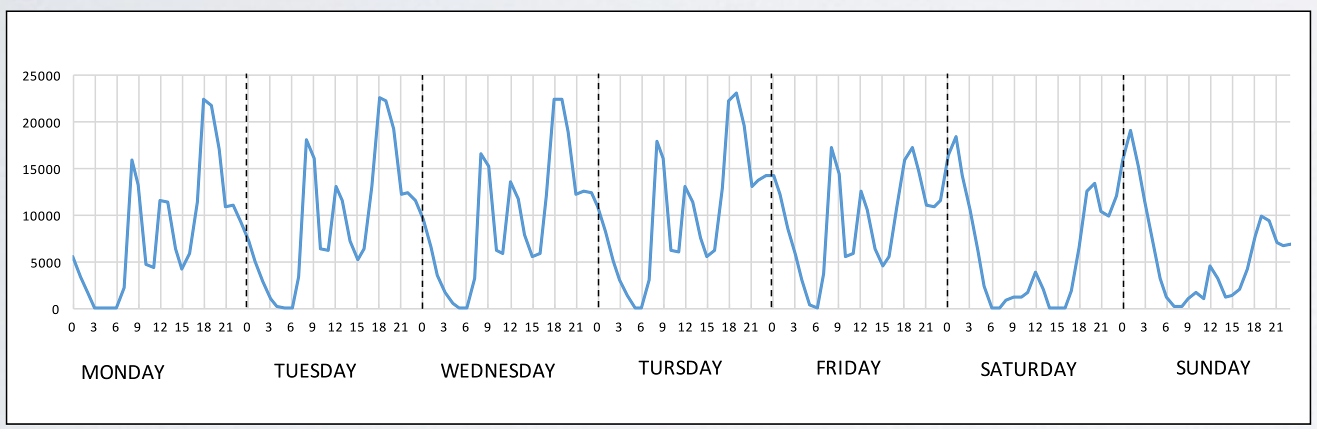
Automatically discovered patterns



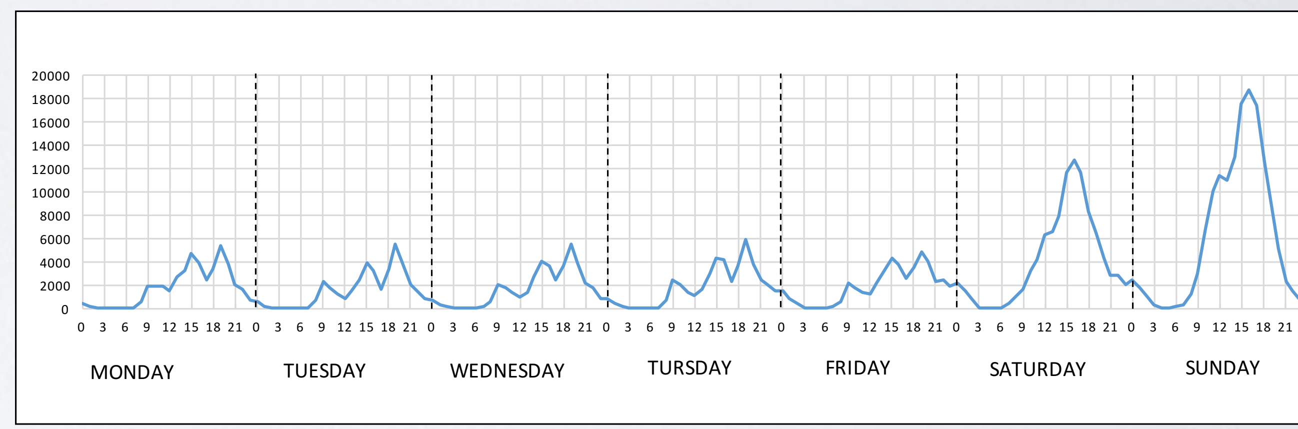
"Commercial" ?



"Work" ?



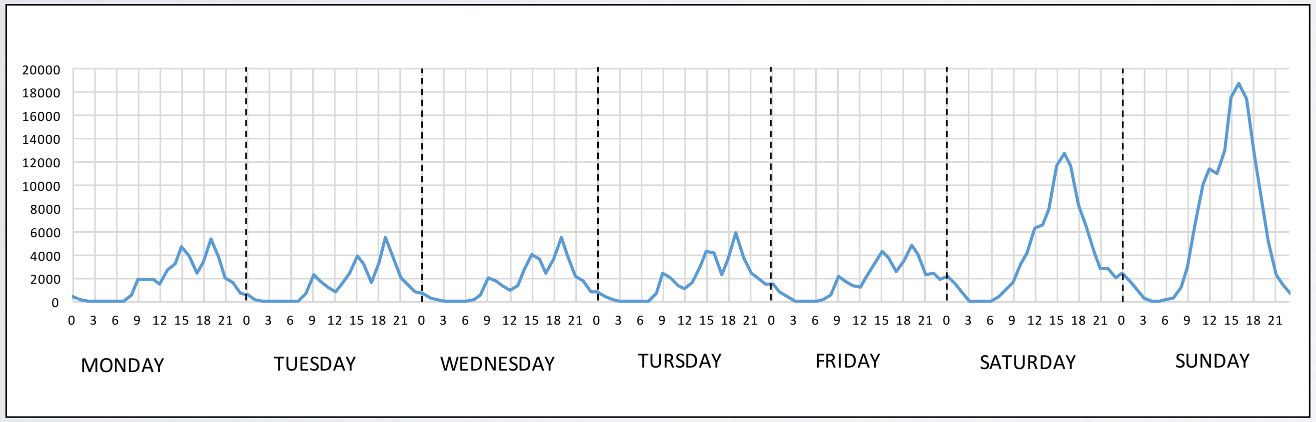
"Bars-Restaurants" ?



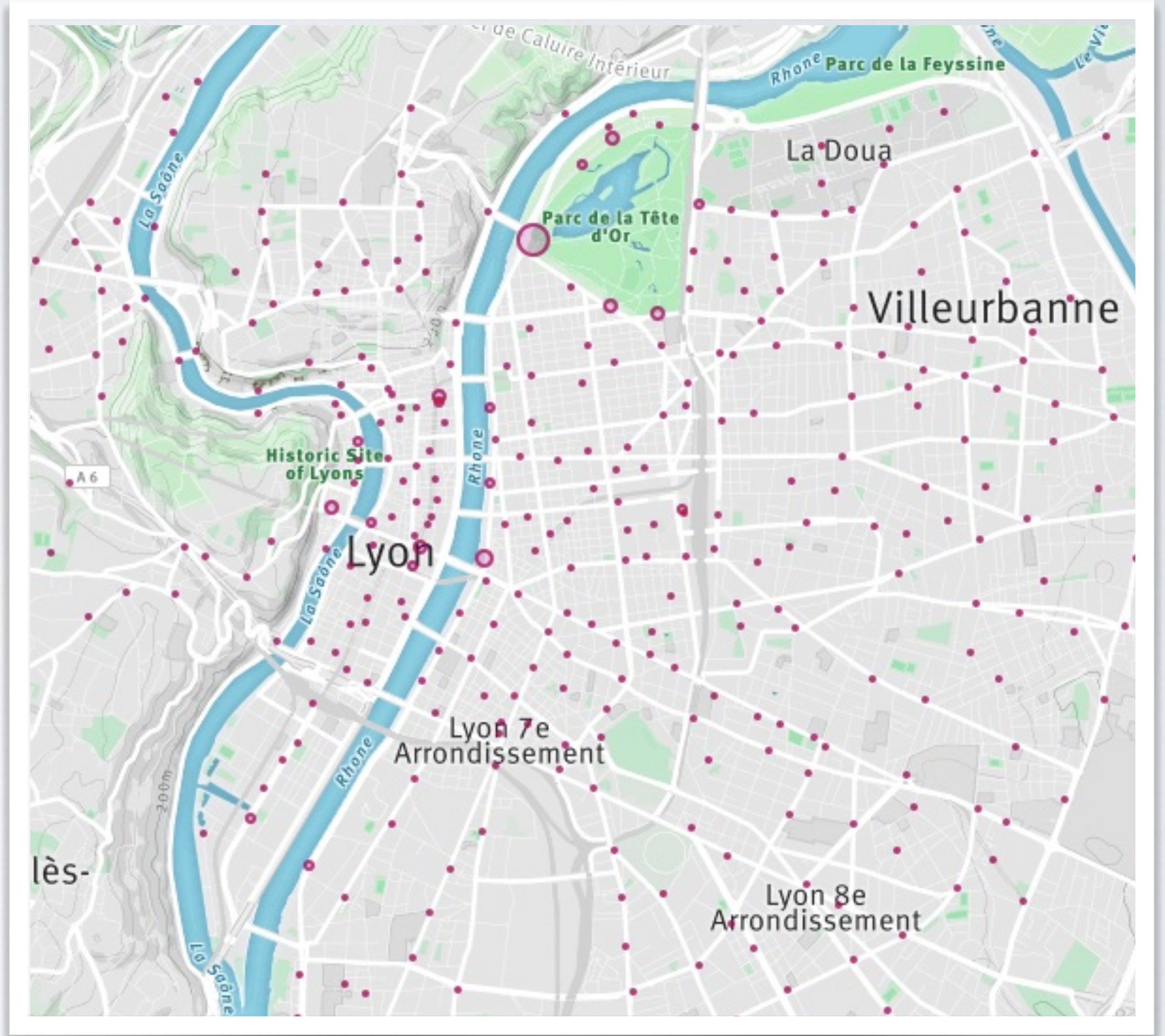
"Leisure" ?

. . .

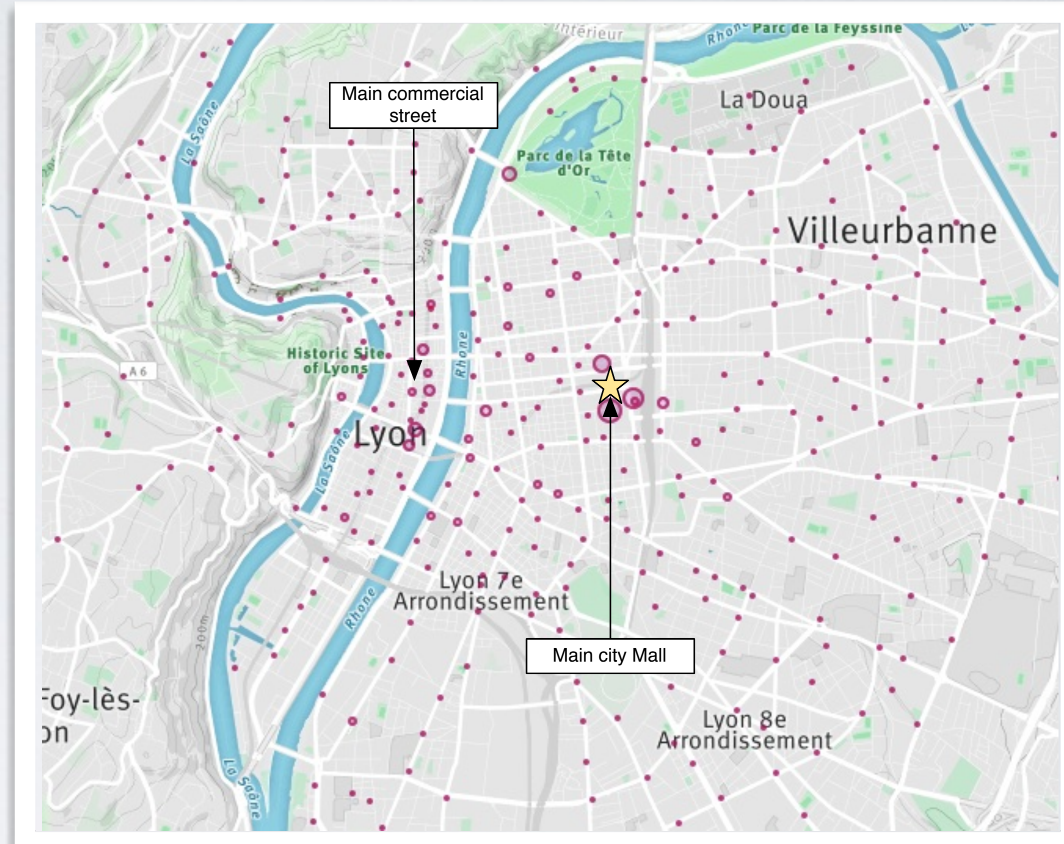
For each pattern, for each station, we have a value
=>Total trips due to this pattern

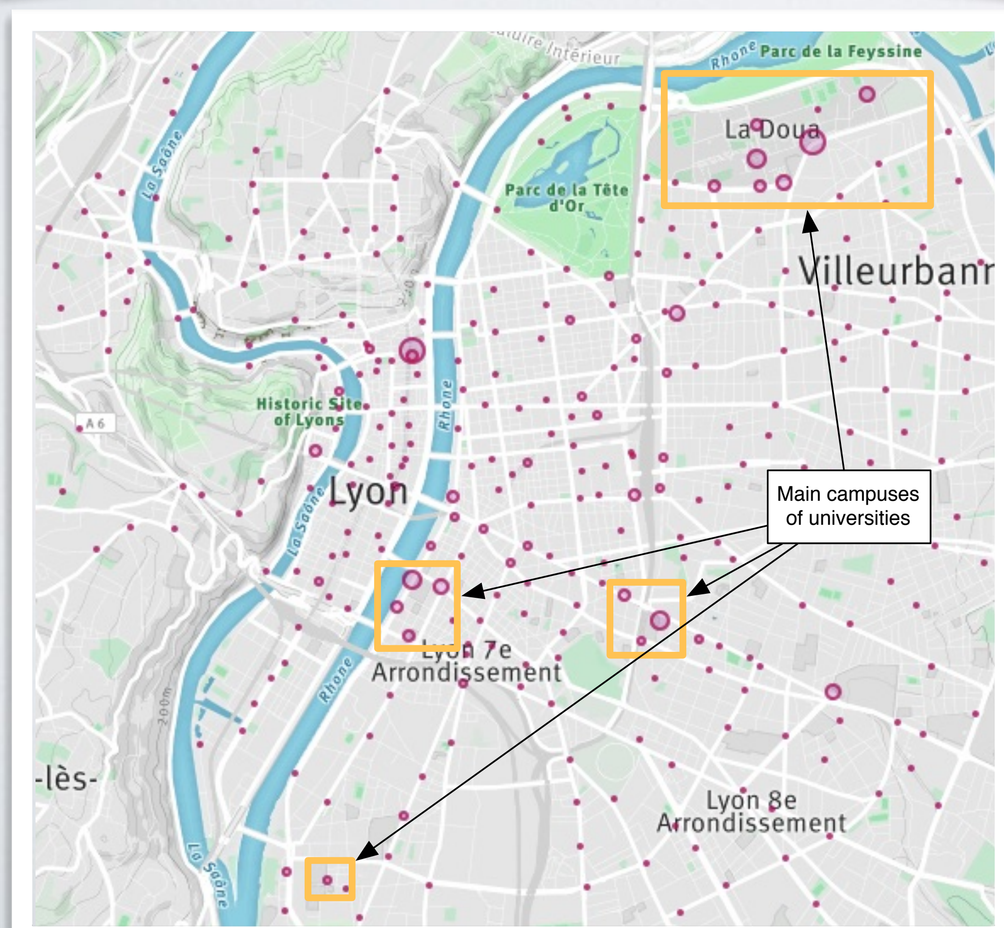


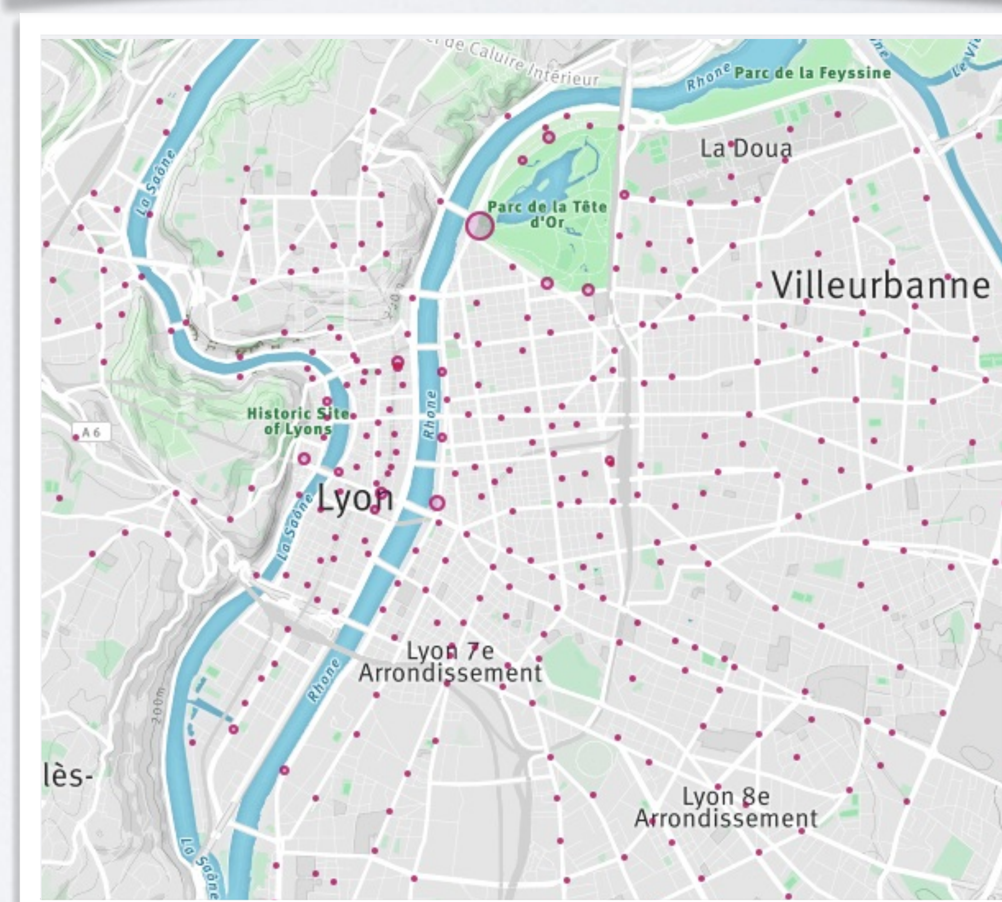
"Leisure" ?




La Doua
Parc de la Tête d'Or
Villeurbanne
Historic Site of Lyons
Lyon
Lyon 7e Arrondissement
Main train station
Lyon 8e Arrondissement


Parc de la Feyssine
La Doua
Main commercial street
Parc de la Tête d'Or
Villeurbanne
Historic Site of Lyons
Lyon
Lyon 7e Arrondissement
Main city Mall
Lyon 8e Arrondissement


Parc de la Feyssine
La Doua
Parc de la Tête d'Or
Villeurbann
Historic Site of Lyons
Lyon
Main campuses of universities
Lyon 7e Arrondissement
Lyon 8e Arrondissement

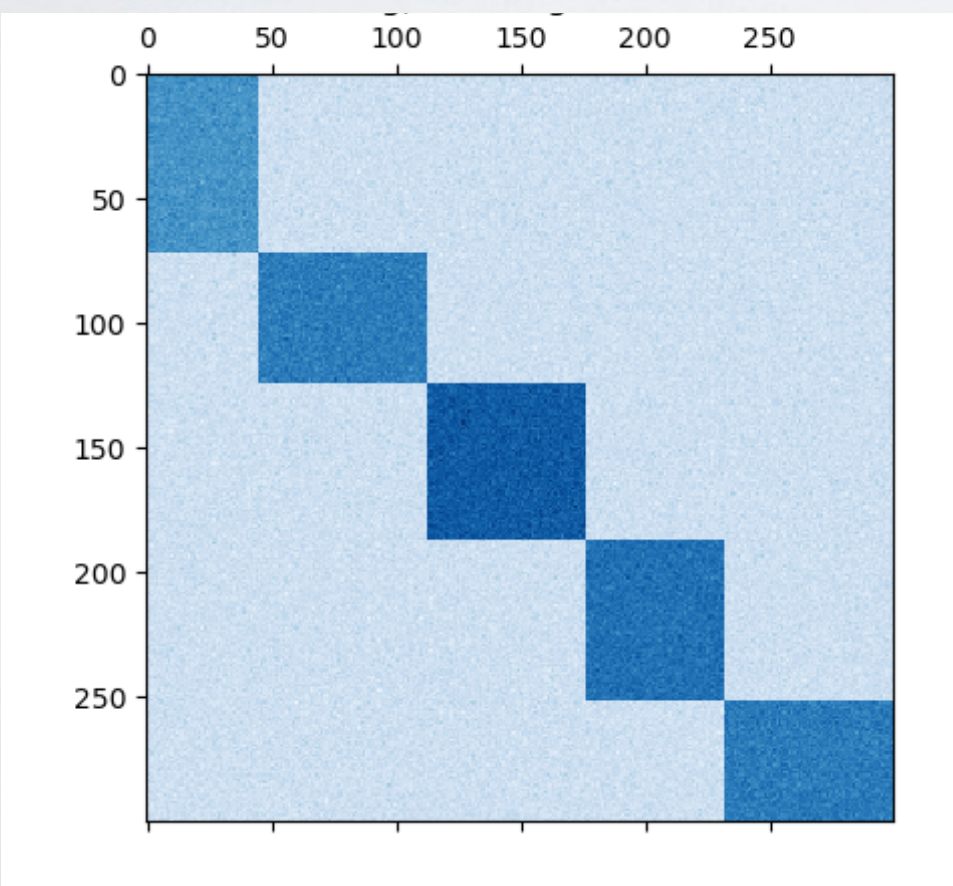

Parc de la Feyssine
La Doua
Parc de la Tête d'Or
Villeurbanne
Historic Site of Lyons
Lyon
Lyon 7e Arrondissement
Lyon 8e Arrondissement

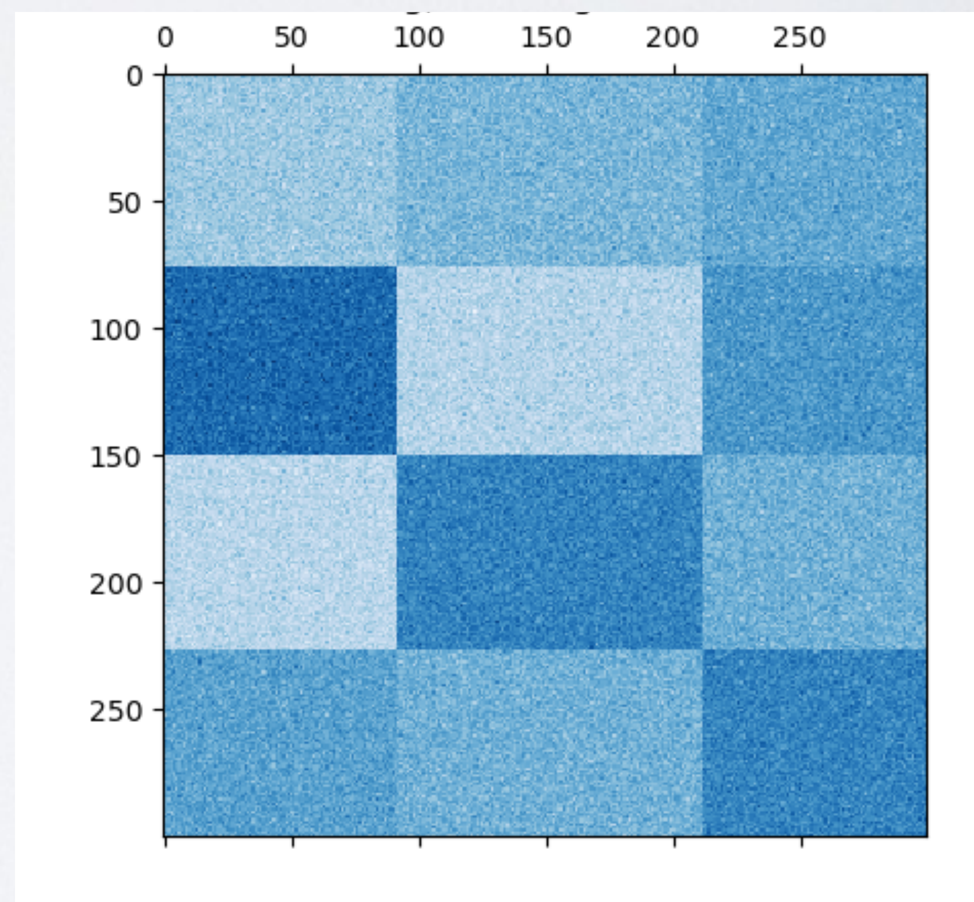
CO-CLUSTERING

Or Bi-clustering, two-mode clustering, block clustering

CO-CLUSTERING

- Objective: Find dense submatrices in a matrix
- Groups of rows that are preferentially related to groups of columns





CO-CLUSTERING

- Various algorithms exist, a simple one for sparse data consists in optimizing a modified version of the modularity on the bipartite graph (user-item)

- $$Q = \sum_{i}^{n} \sum_{j}^{d} A_{ij} - \frac{k_i k_j}{|A|} \delta_{ij}$$
    - With A the matrix to co-cluster, dimension $n \times d$
    - k_i the weighted degree(strength) of i
    - δ_{ij}=1 if i, j belong to the same co-cluster
    - $|A|$ sum of all values in the matrix

https://dl.acm.org/doi/pdf/10.1145/2806416.2806639

CO-CLUSTERING

- Co-cluster make natural sense in user-item matrices
  - Group of people who like the same type of products, and products liked by the same people
- Co-clustering can be used to improve recommender systems
  - To improve scalability, one can compute co-cluster first, and then use only users/items in the same co-cluster for recommendation
  - It can also improve precision: remove the effect of most popular items, that tend to be recommended to everyone