MATRIX-FACTORIZATION
RECOMMENDER-SYSTEMS
BI-CLUSTERING

LATENT FACTORS

- A popular problem in Data Mining
- Given two types of data
  - Users and Items (Client buying, interacting with content in social media…)
  - Locations and Dates (T°, mortality in cities along week/year…)
- Unsupervised task
  - How to best reconstruct the data
  - By assigning a "latent variable" to each item

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
  - …

CONTENT-BASED

- **Content-based recommendation**
  - We describe all our items using features
    - Movies genre, length, age rate, topics…
    - Objects 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 medias, but also Amazon, YouTube..

COLLABORATIVE FILTERING

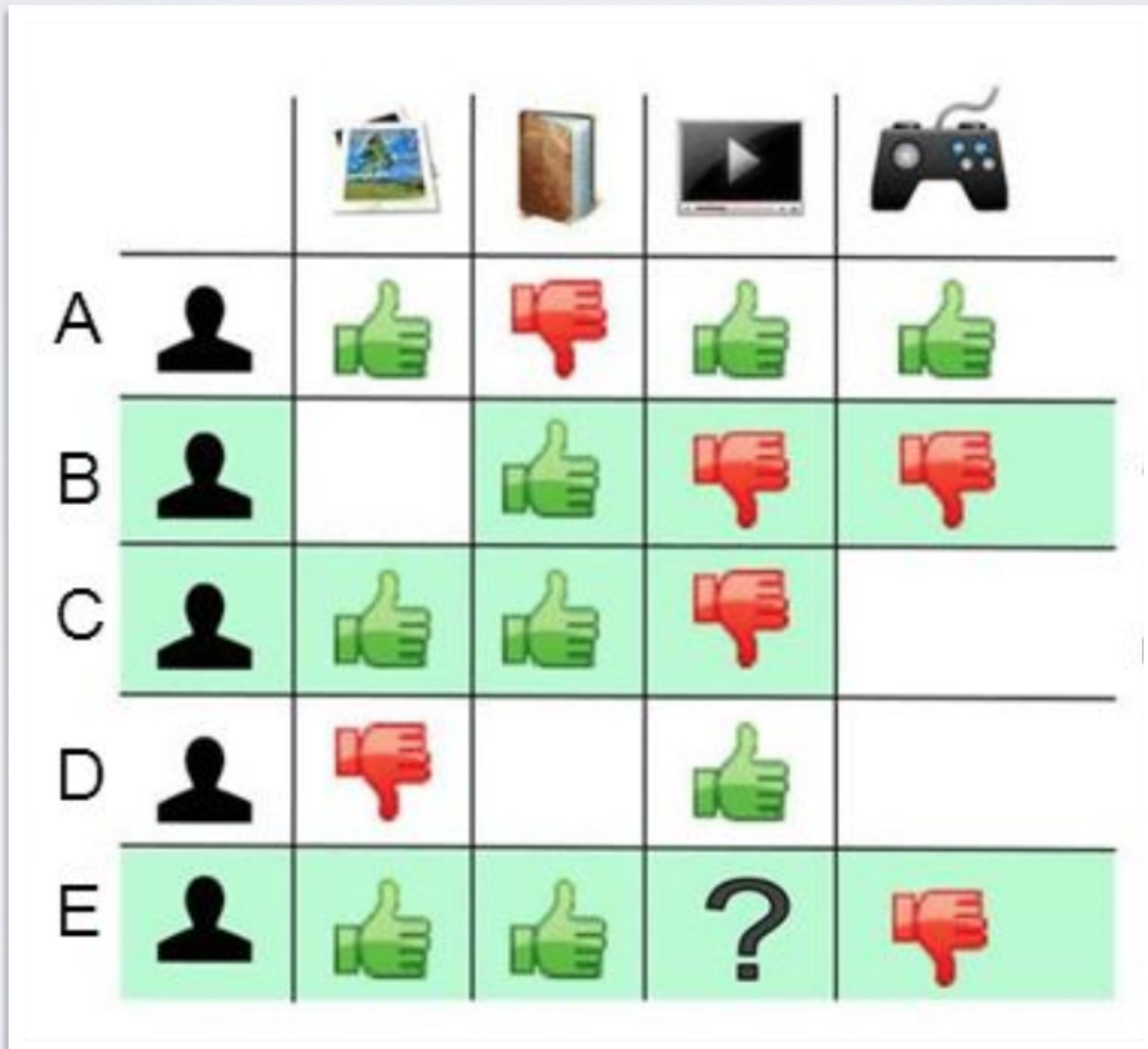
- Solution: **Collaborative filtering**

- 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 a 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)

- Users can have different labelling 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 never give maximal note, while others use only the maximal note

DATA COMPLEXITY

- User note diversity => Normalize/Standardize scores for each user

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

- Considering both aspects: subtracting a *baseline*
  - We estimate the baseline score (u, i) based on 2 constants, 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
    - e.g., minimize by gradient descent a regularized baseline
    - $\sum_{r_{ui} \in R_{train}} \left(r_{ui} - (\mu + b_u + b_i)\right)^2 + \lambda \left(b_u^2 + b_i^2\right).$
    - μ: average note for a random user on a random item

USER-BASED KNN

USER-BASED KNN

- KNN: K-Nearest-Neighbors
  - Simple yet powerful method popular in <u>classification</u> 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

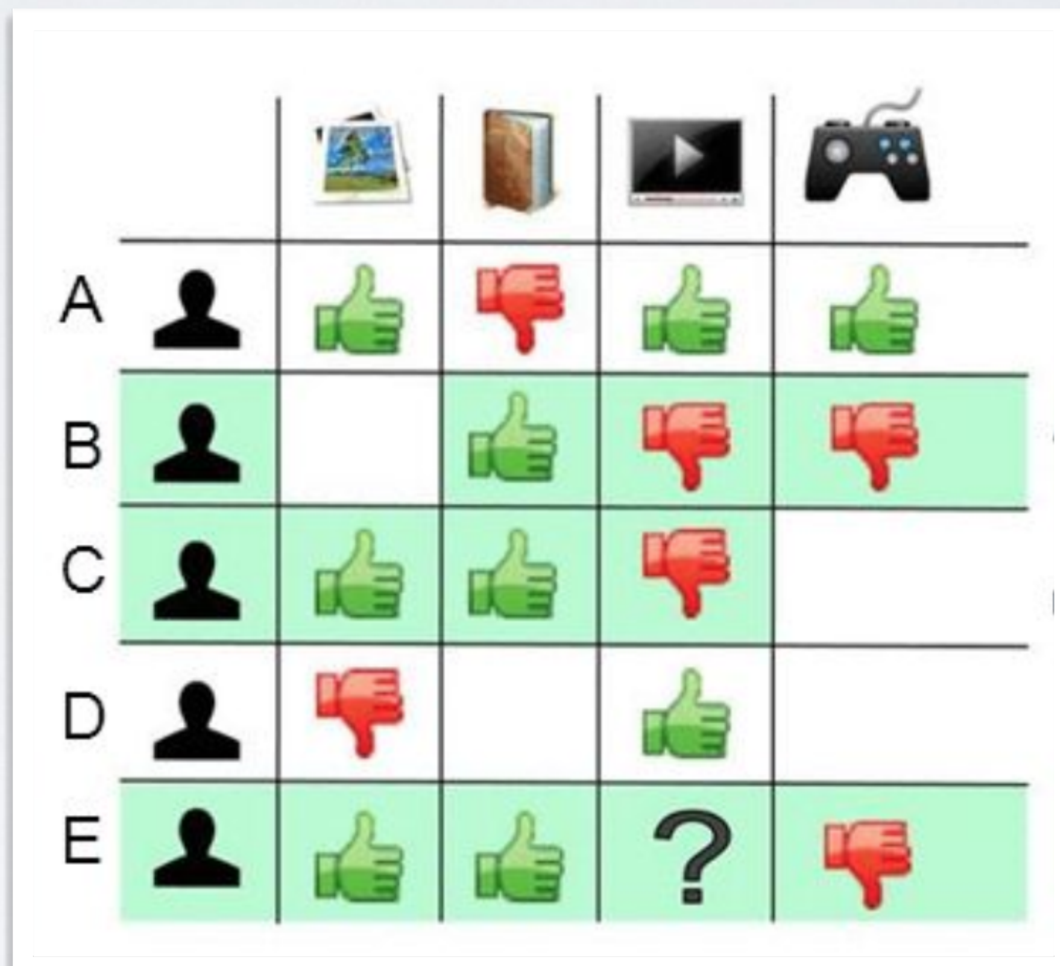
Similarity to E

-1

2

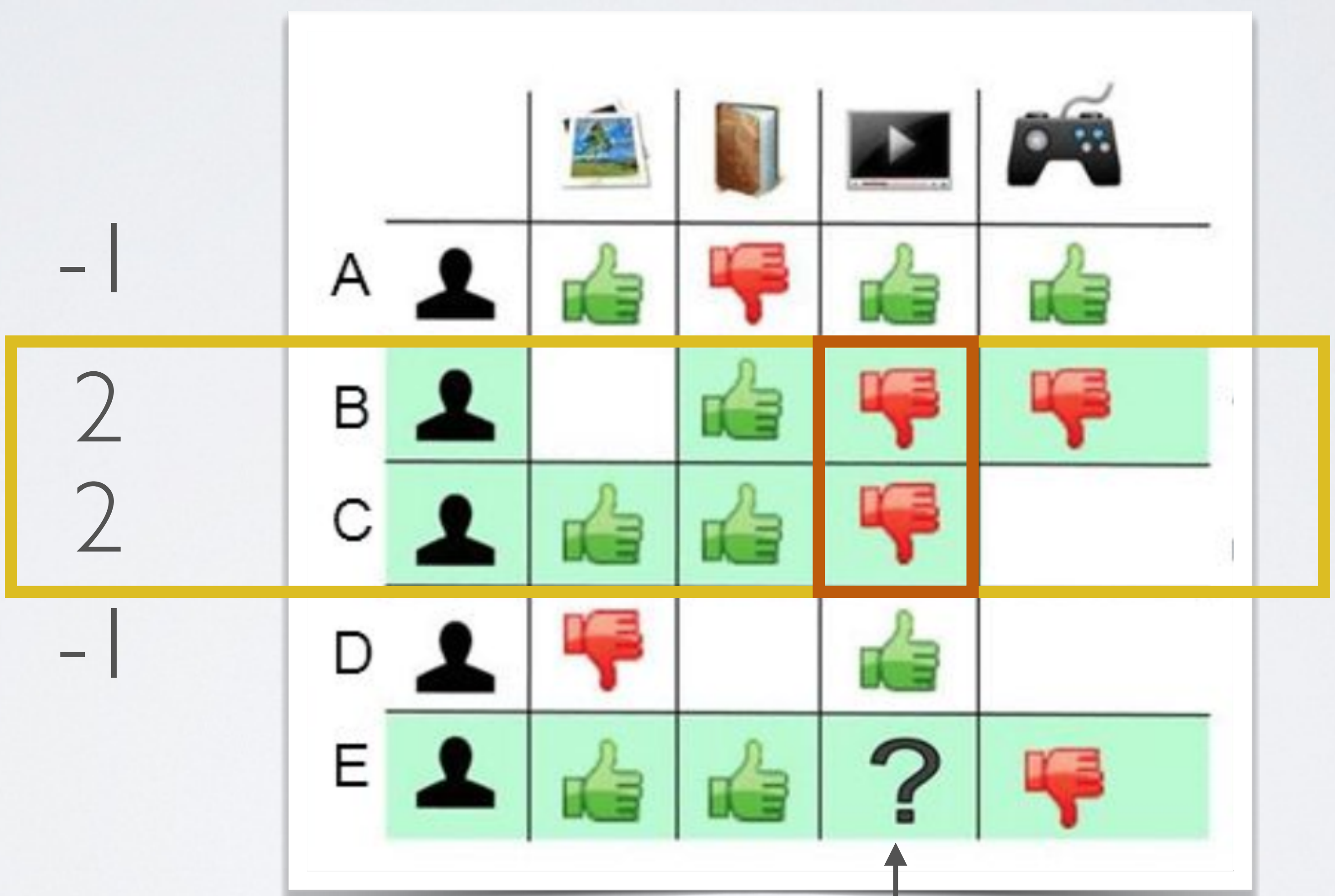
2

-1

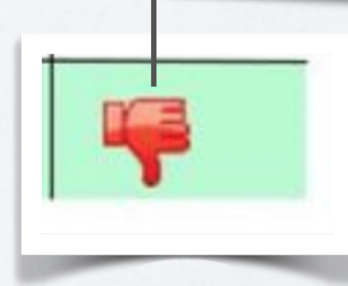


USER-BASED KNN

Similarity to E



(2*-1)/2=-1



SIMILARITY

- How to compute the similarity between users ?
  - Euclidean distance => Poor results
    - 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 ?
    - Now users with many likes are similar to everyone

- Solution:
  - (Binary) Jaccard Similarity => | likes(u&v) | / (union like)
  - (Notes) MSD=>Means Squared Difference <u>when both notes present</u>
  - (Binary & Notes) Cosine Similarity

SIMILARITY

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

For binary:

$$\frac{|\text{likes}(u \,\&\, v)|}{\sqrt{|\text{likes}(u)|}\sqrt{|\text{likes}(v)|}}$$

Similar Principle than Jaccard Coefficient

ITEM-BASED
COLLABORATIVE FILTERING

ITEM-BASED

- User-based collaborative filtering has weaknesses in practice
  - Users with little info will have neighbors with little info too
    - =>We will learn based on few info
    - Imagine you liked movies M1 and M2. The 20 most similar users will like exactly M1 and M2, maybe 1 movie more.
  - Users change a lot =>Need to recompute KNN on whole database very frequently

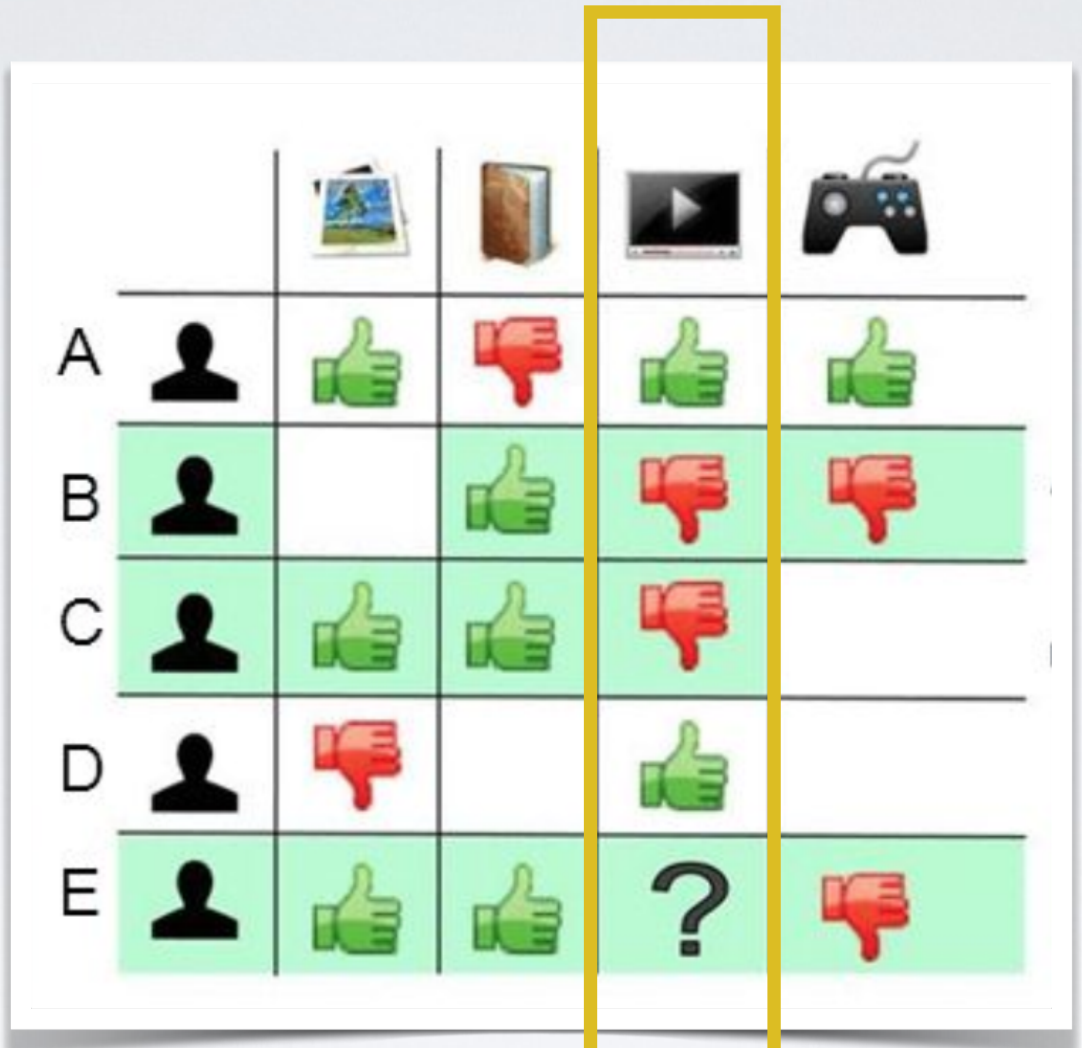
- => Move to Item-based Collaborative filtering

ITEM-BASED

- We want to evaluate the interest of (u,i)
  - 1)For each item x liked by u
    - Compute the similarity between x and i
  - 2)(u,i) is the average similarities (x,i) for x liked by u

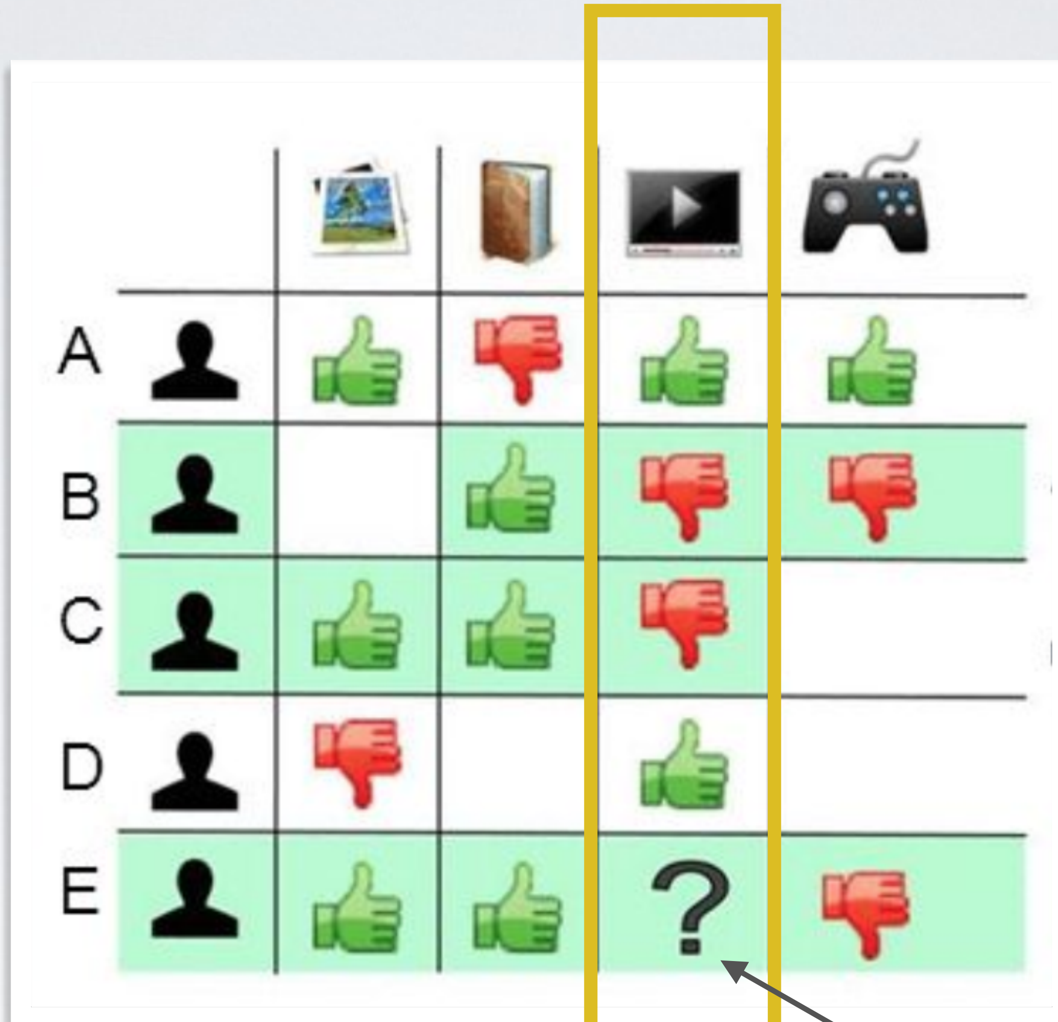
- We compute score (u,i) for every unknown item

ITEM-BASED



-1 -3 2

ITEM-BASED



-1 -3 2

=(1*(-1)+1*(-3)-1*2)/3=>-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

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
- Yet, one new popular approach attracted lot of attention: SVD
  - /!\ Singular Value Decomposition(SVD) is a classic linear algebra matrix decomposition. But in recommendation literature, SVD is also the name of an algorithm related but different to the original SVD.

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

MATRIX FACTORIZATION

- **Matrix Factorization** is a name given to a general approach of data mining
  - 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

- If the dimension of A is $X \times Y$
  - Then $U = > X \times D$, $V = > D \times Y$
    - With D a parameter, corresponding to a number of *latent variables*
  - The process is a type of dimensionality reduction

MATRIX FACTORIZATION

	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
	✓		✓	✓	
		✓			✓
	✓	✓	✓		
				✓	✓

≈

.9	-1	1	1	-.9
-.2	-.8	-1	.9	1

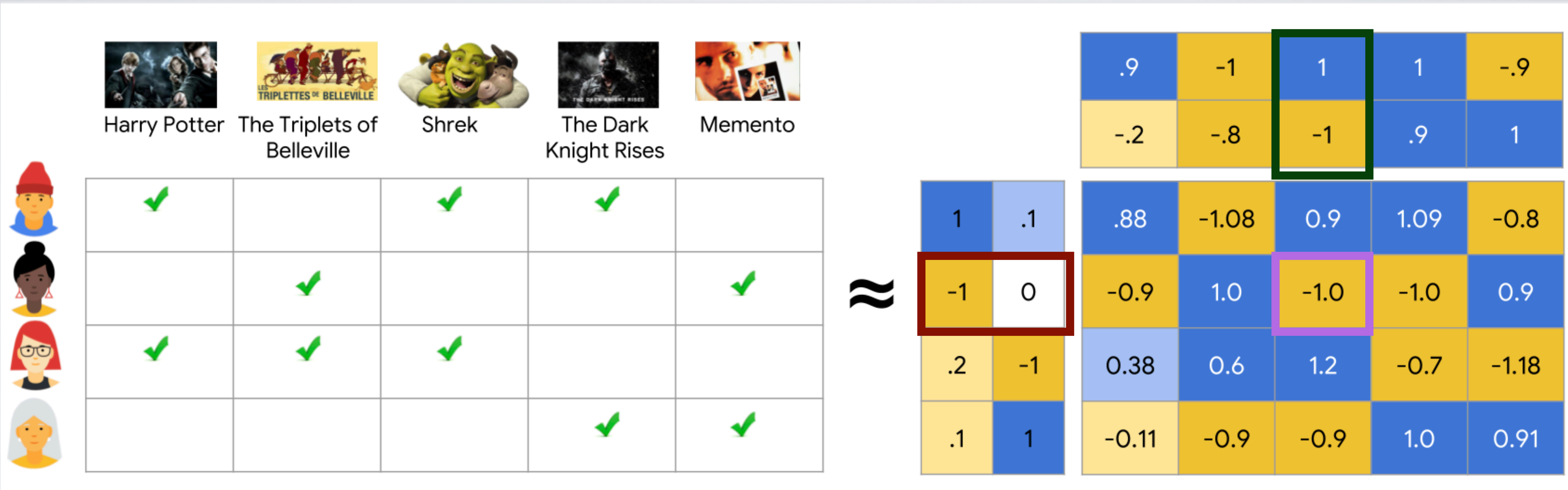
1	.1
-1	0
.2	-1
.1	1

.88	-1.08	0.9	1.09	-0.8
-0.9	1.0	-1.0	-1.0	0.9
0.38	0.6	1.2	-0.7	-1.18
-0.11	-0.9	-0.9	1.0	0.91

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

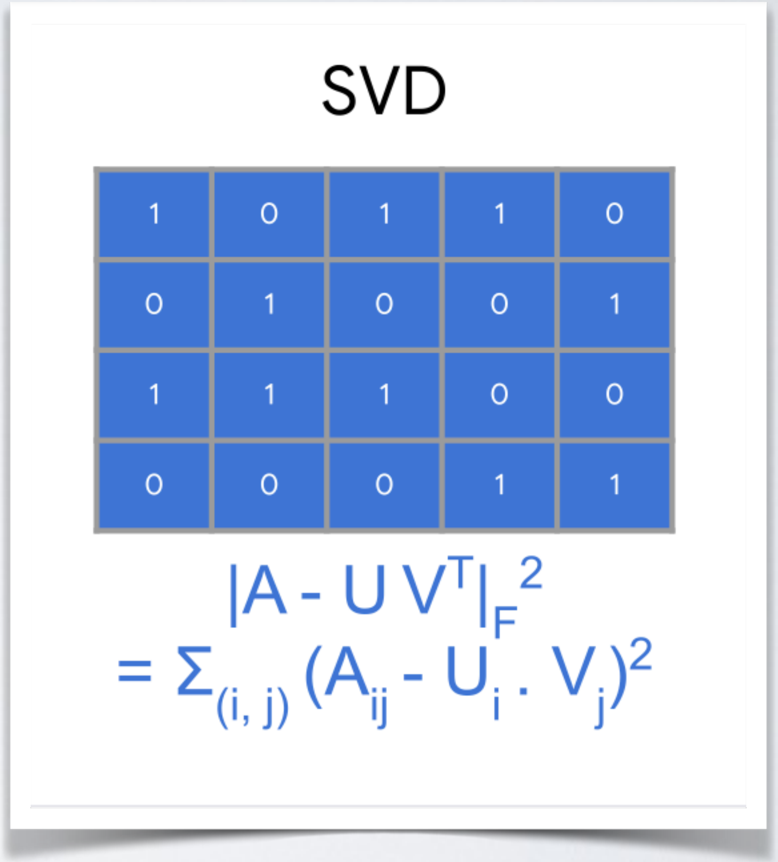
MATRIX FACTORIZATION

- As with word embedding approaches (word2vec, etc.), 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

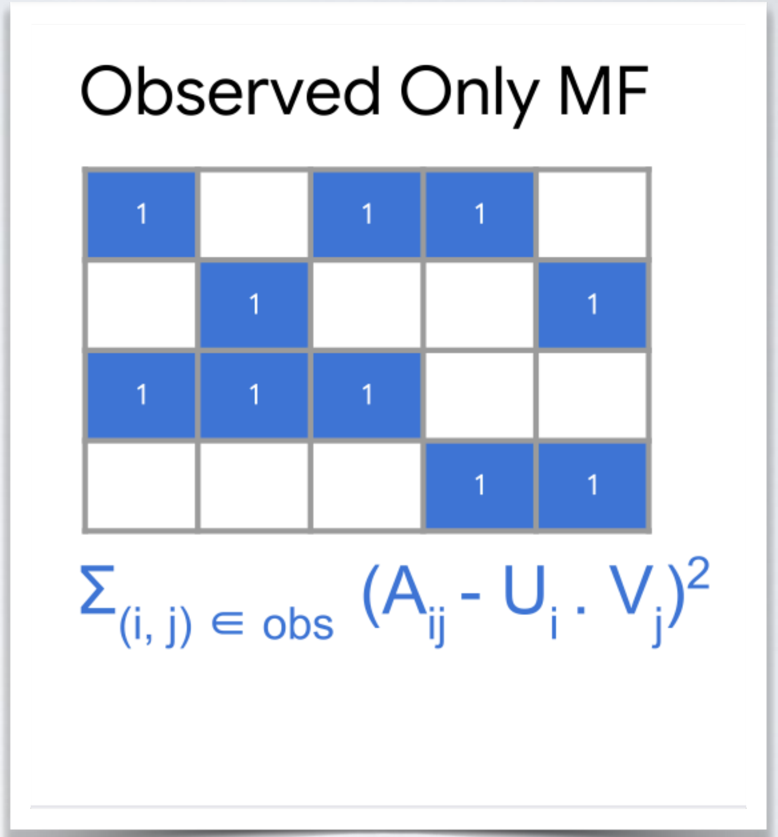
OBJECTIVE FUNCTION

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

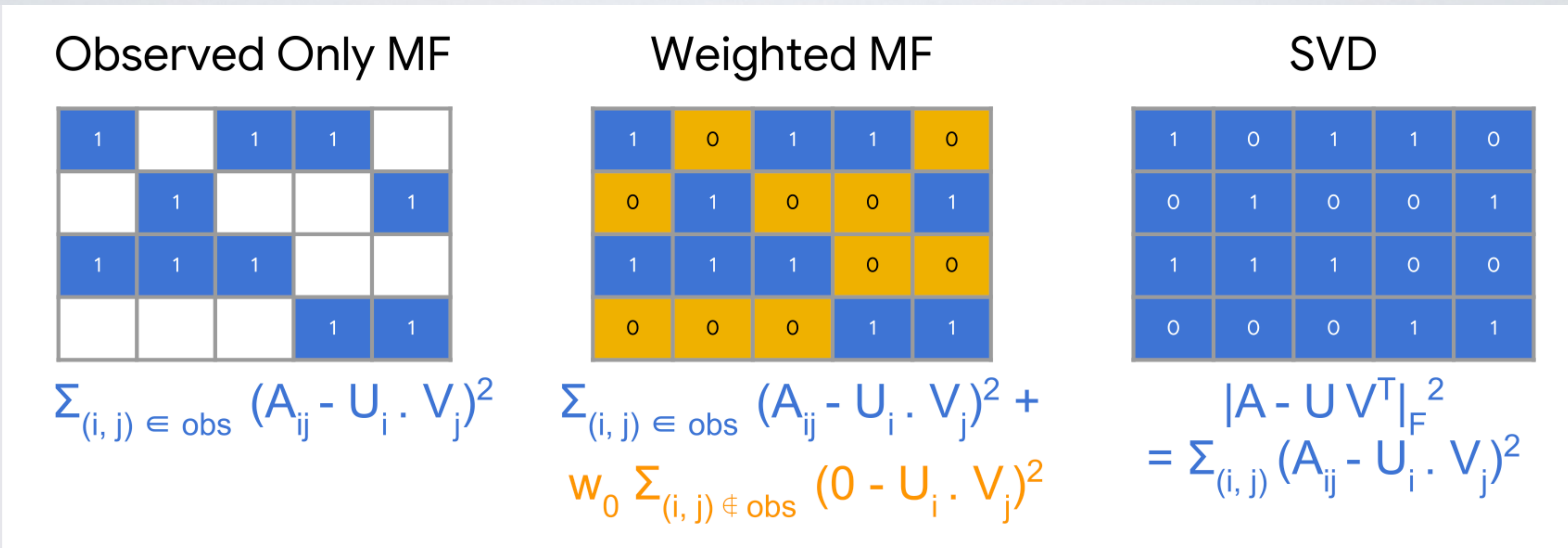


OBJECTIVE FUNCTION

- The recommendation based Matrix Factorization has an adapted loss,
  - **Computed only on non-zero values**



OBJECTIVE FUNCTION



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

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

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

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:
  - $$\sum_{r_{ui} \in obs} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda \left(||q_i||^2 + ||p_u||^2 \right)$$
    - q_i, p_u are latent vectors
  - λ controls the strength of the regularization

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
    - => If most users give good marks to movie M1, and user U1 tend to always give maximal scores to movies they rate, the fact that (U1,M1)=maximal note is "expected"

- The objective to solve becomes:
  - $$\sum_{r_{ui} \in obs} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right)$$
  - With b_i and b_u representing items and users expected scores, respectively

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

.9	-1	1	1	-.9
-.2	-.8	-1	.9	1

1	.1
-1	0
.2	-1
.1	1

.88	-1.08	0.9	1.09	-0.8
-0.9	1.0	-1.0	-1.0	0.9
0.38	0.6	1.2	-0.7	-1.18
-0.11	-0.9	-0.9	1.0	0.91

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…

NEW USER

- If a new user requests a recommendation, the complexity to provide one depends on the method
  - User based=>Compute distance to all other users
    - Then direct answer for all items
  - Item based=>Precomputed distances betweeen all items
    - Naive approach, need to compute for all candidate items, but in reality, faster tricks
    - e.g., Find items that are "close" to the ones liked by that user
  - Matrix Factorization
    - In theory, not possible to make recommendation to a new user without recomputing everything
    - In practice, an approximation can be obtained quickly, doing 1 step of the Alternating Least Square: we consider the item latent matrix fixed, updating the user matrix. Similar in nature to solving a linear regression

EVALUATION OF RECOMMENDER SYSTEMS

EVALUATION

- Recommendation evaluation use similar quality scores as supervised machine learning evaluation
  - RMSE, Precision@k, AUC, etc.
- The specificity of recommender systems is the way the train and test sets are built
  - General principle: For one *test* user,
    - We **show** part of their scores/votes to the trained recommender
    - We **hide** part of them, to use as ground truth
  - The problem is thus either:
    - A regression: how accurately do we predict the scores of hidden items
    - A classification: how many of the positive items in the test set do we recommend? Or, more realistically, AUC=Do we assign high scores to positive items?

EVALUATION

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

- Hiding users
  - Rarer, but more realistic
    - 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 on other users, considered "new"

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

OTHER RECOMMENDATION QUALITY CRITERIA

- Diversity of recommendation
  - e.g., 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, or information entropy…
- Personalization
  - e.g., average cosine distance between users recommendation

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

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

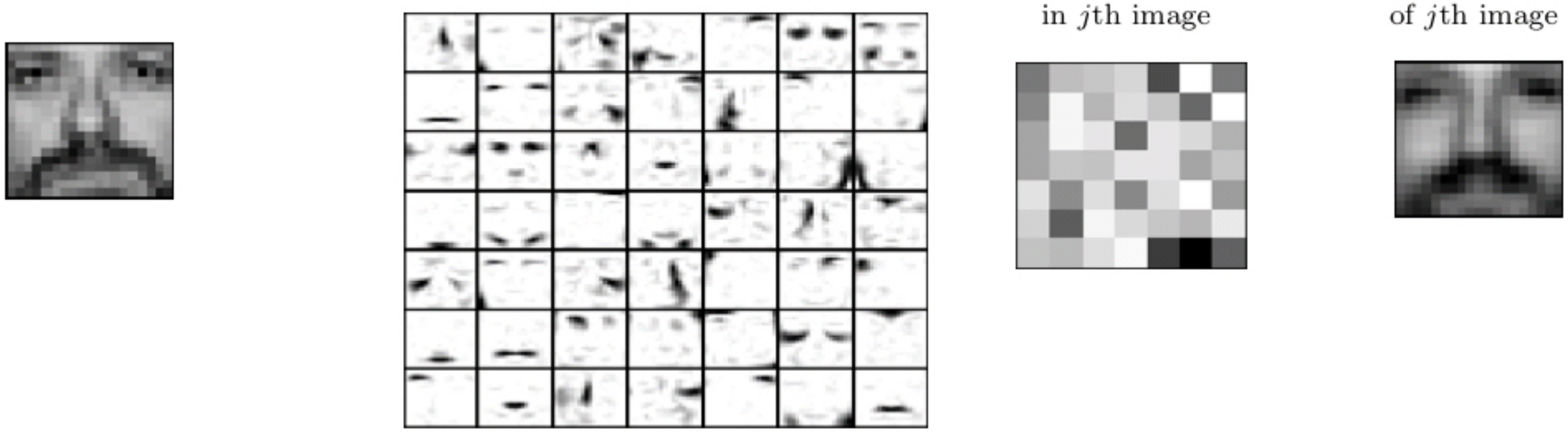


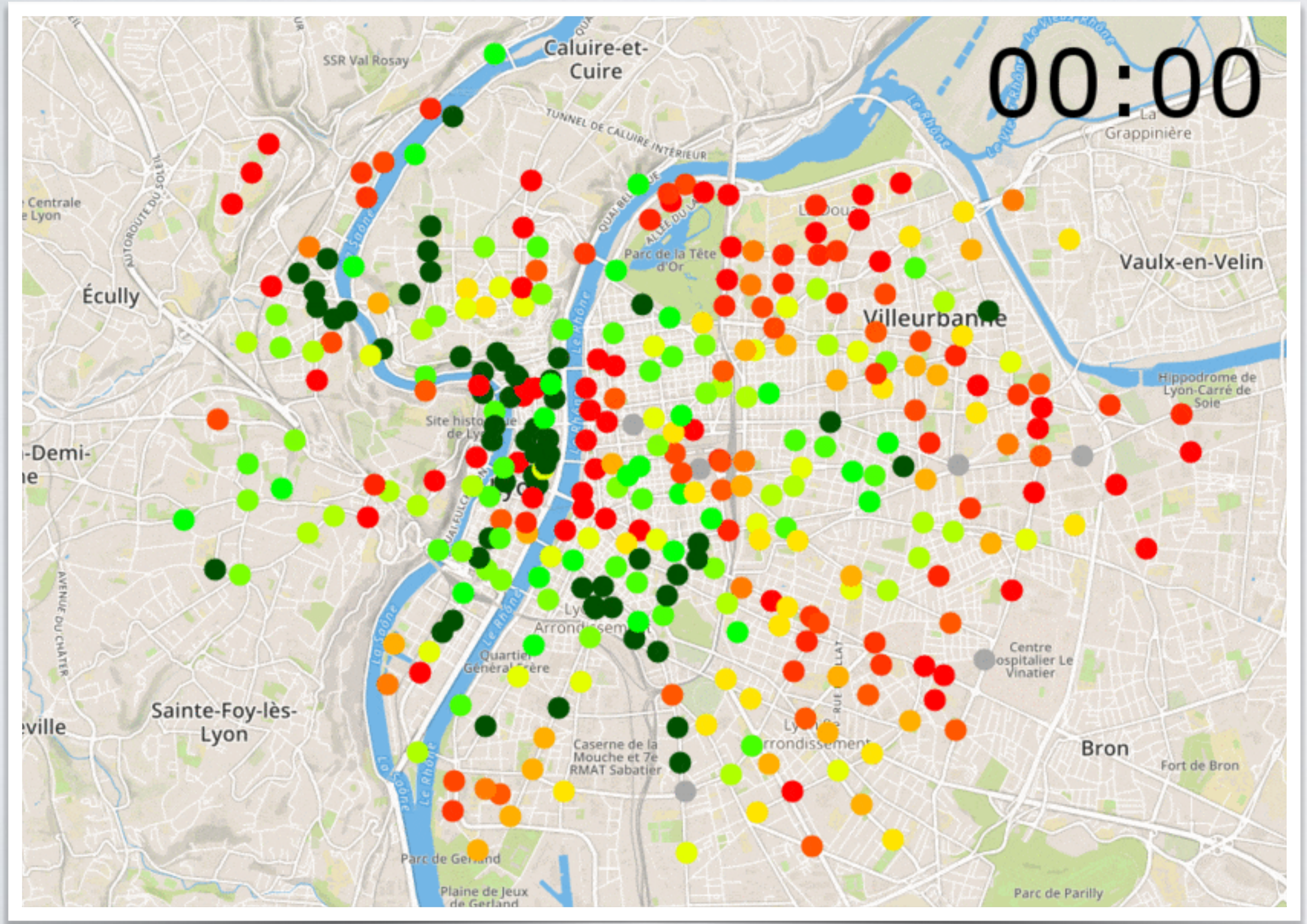
Figure 1: Decomposition of the CBCL face database, MIT Center For Biological and Computation Learning (2429 gray-level 19-by-19 pixels images) using $r = 49$ as in [79].

BICYCLE SHARING SYSTEMS

Docking stations

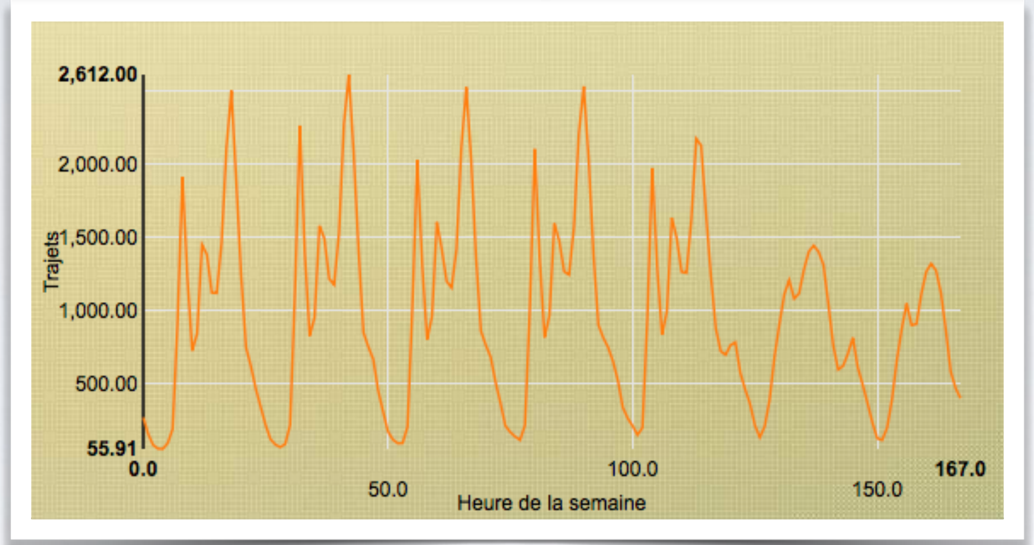
Bicycle trips

DATA

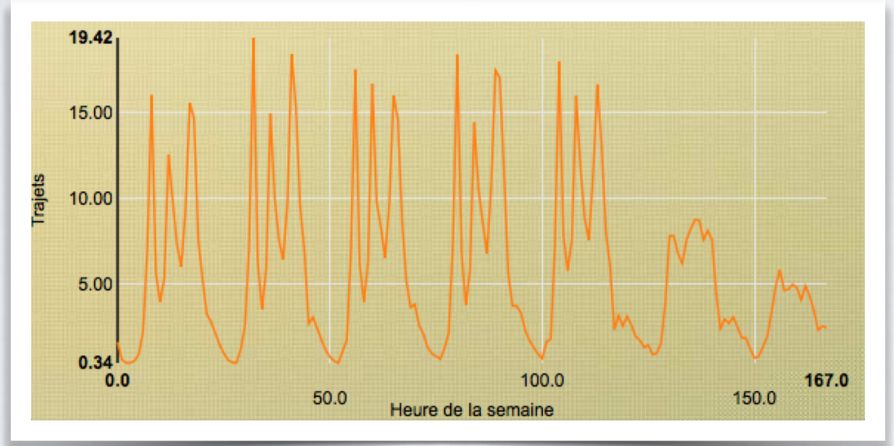


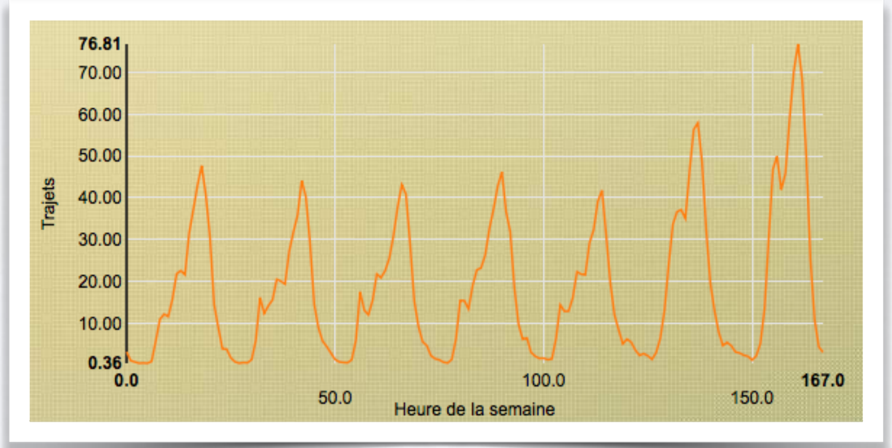
Red: empty

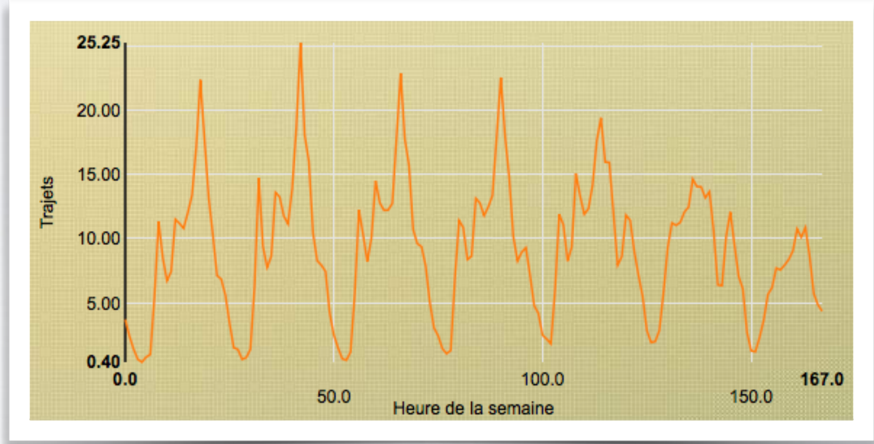
Green: full



Cumulated







Part Dieu

Tête d’or

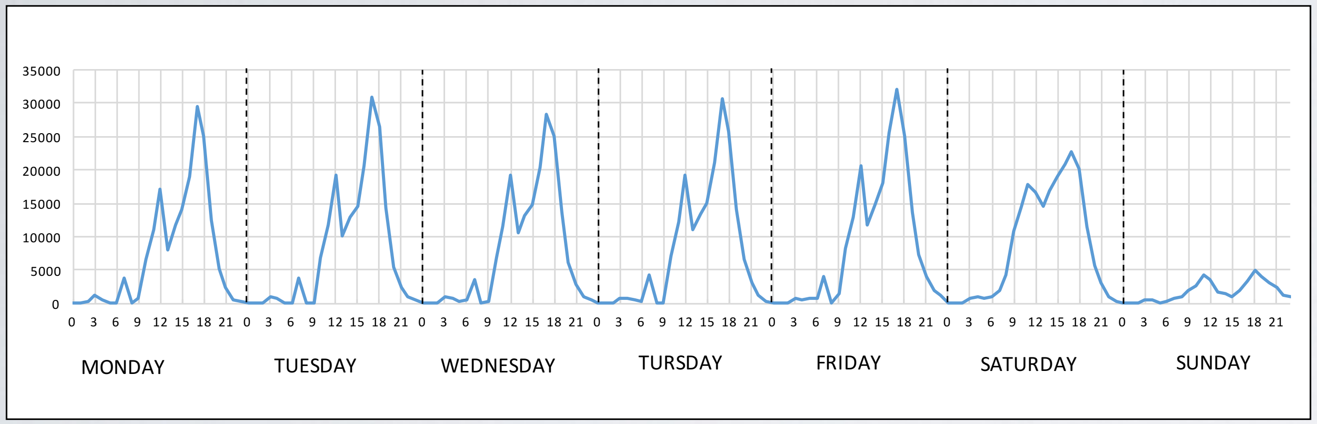
Guillotière

Hours of the typical week

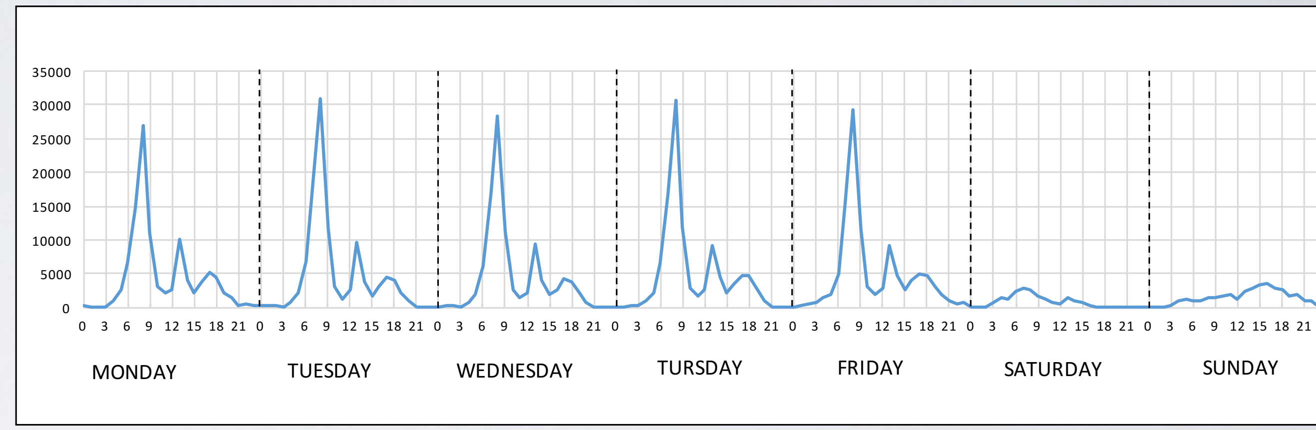
Entities (station)

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

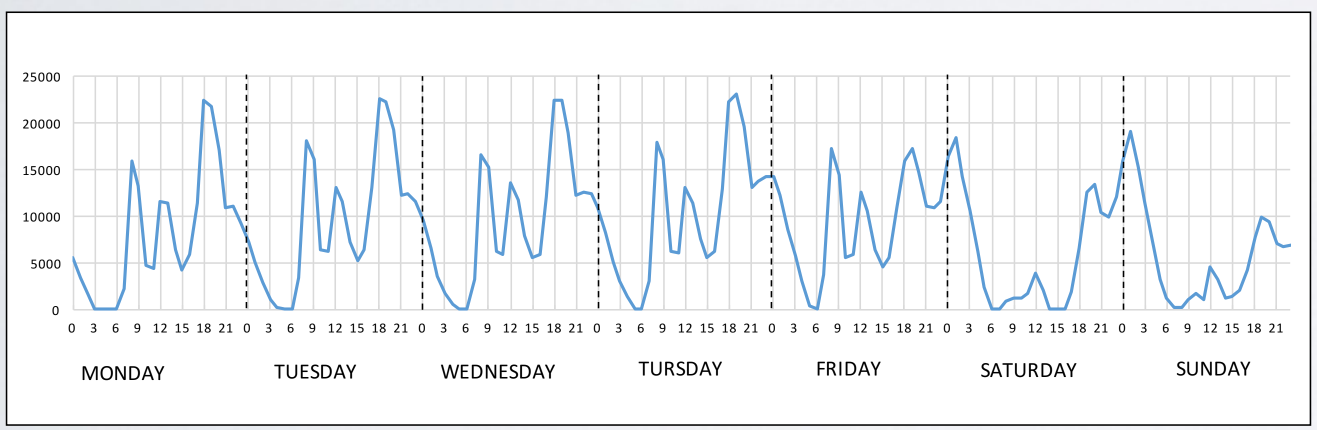
Automatically discovered patterns



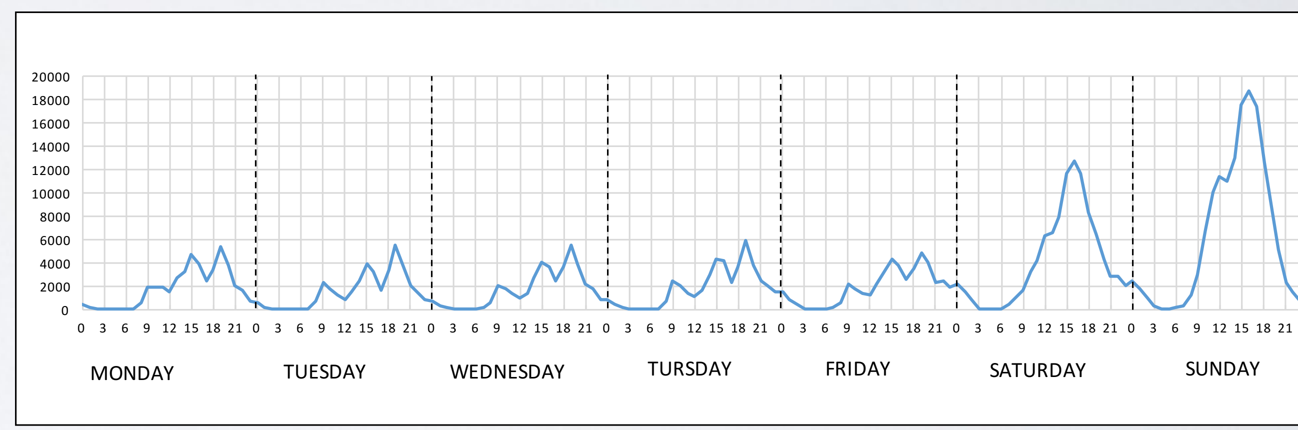
“Commercial” ?



“Work” ?



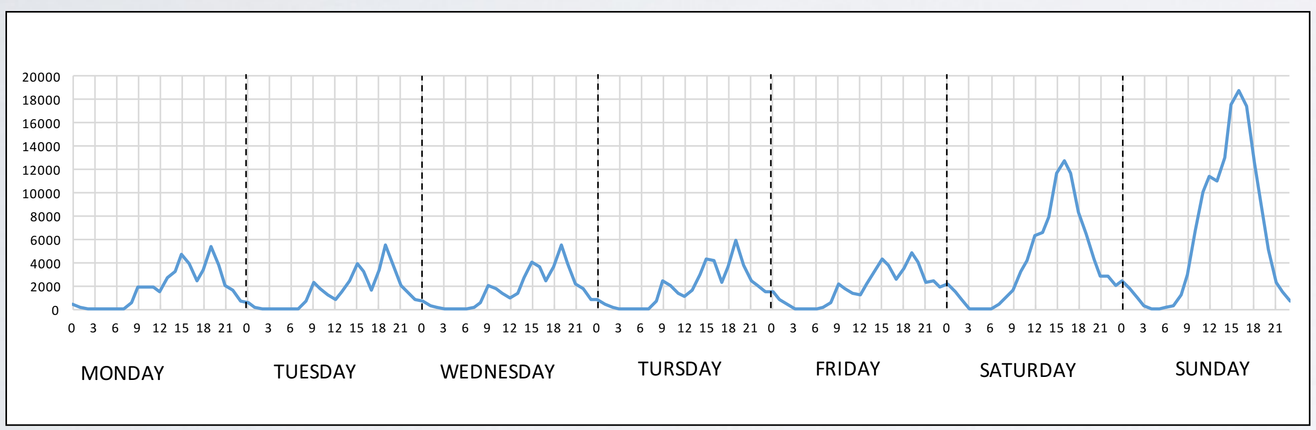
“Bars-Restaurants” ?



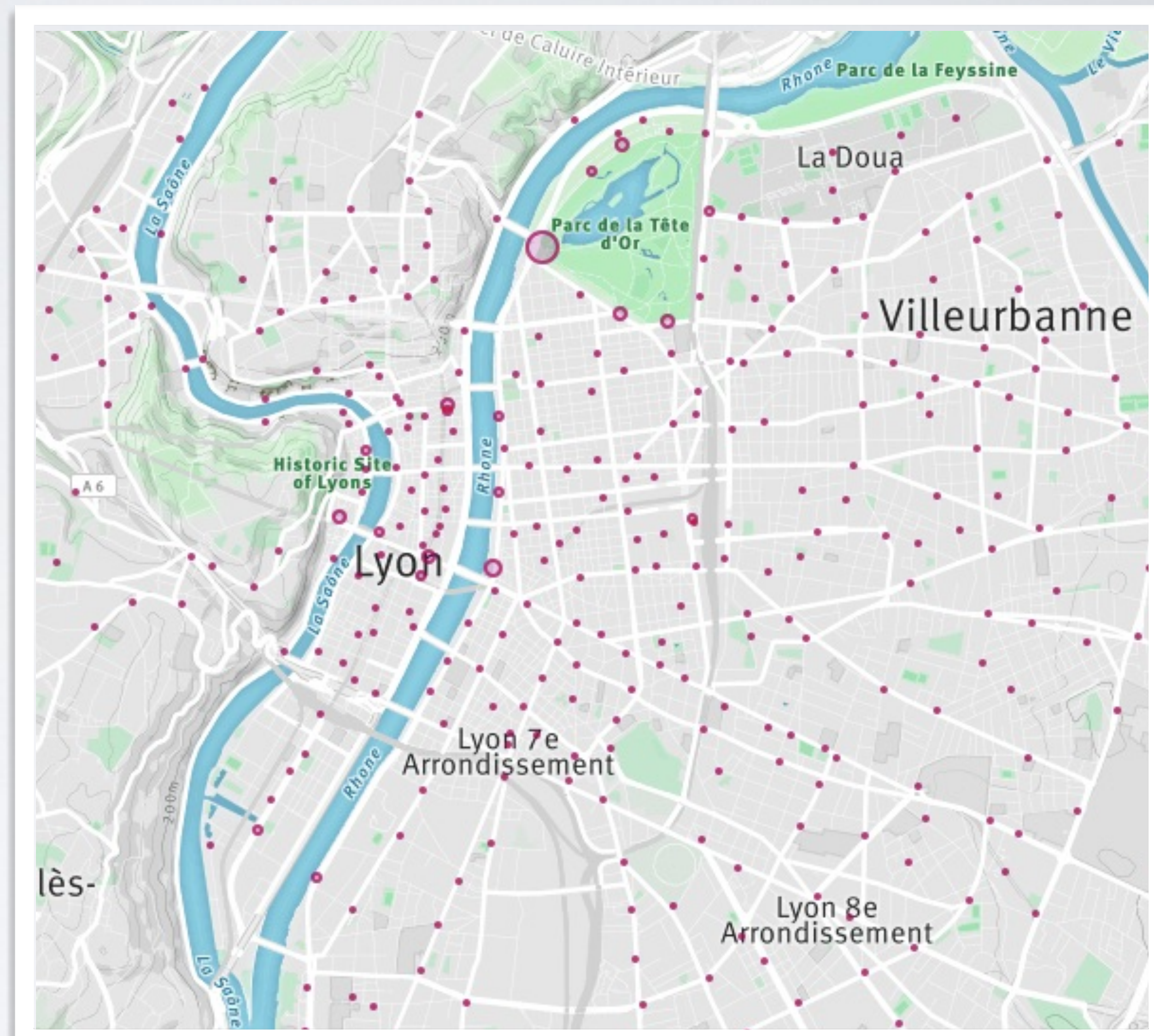
“Leisure” ?

…

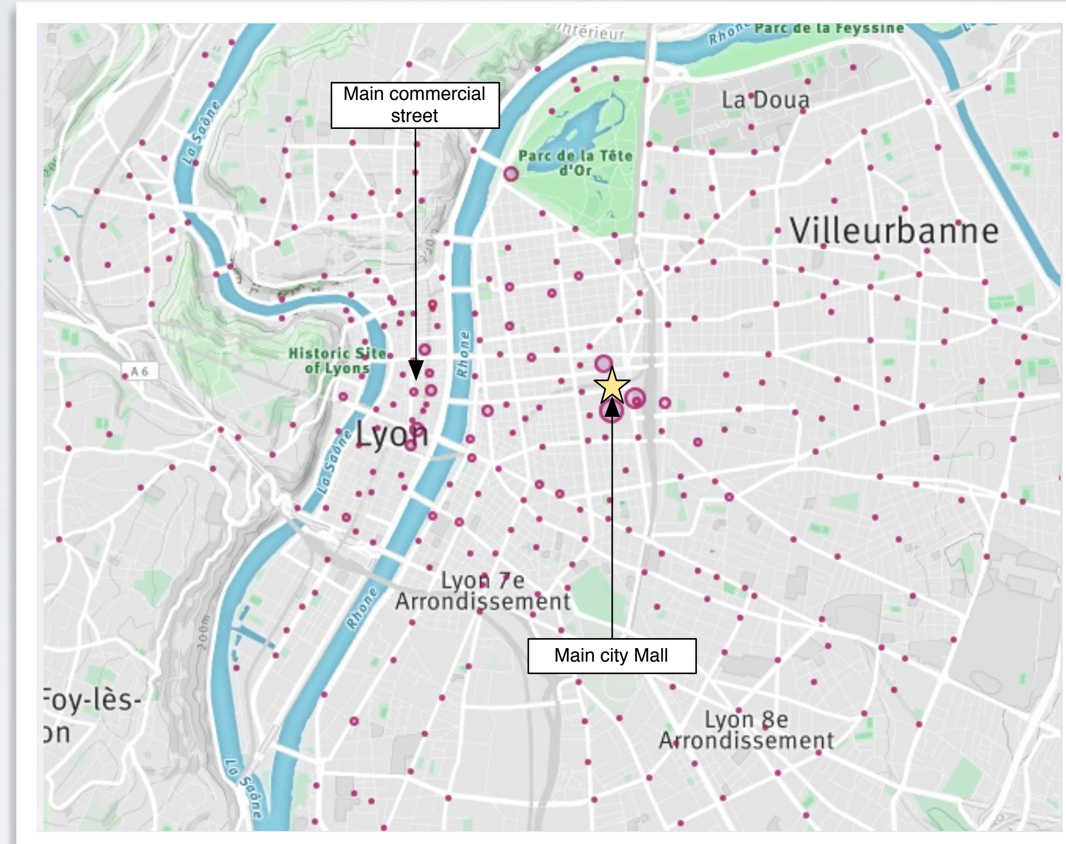
For each pattern, for each station,
we have a value
=>Total trips due to this pattern

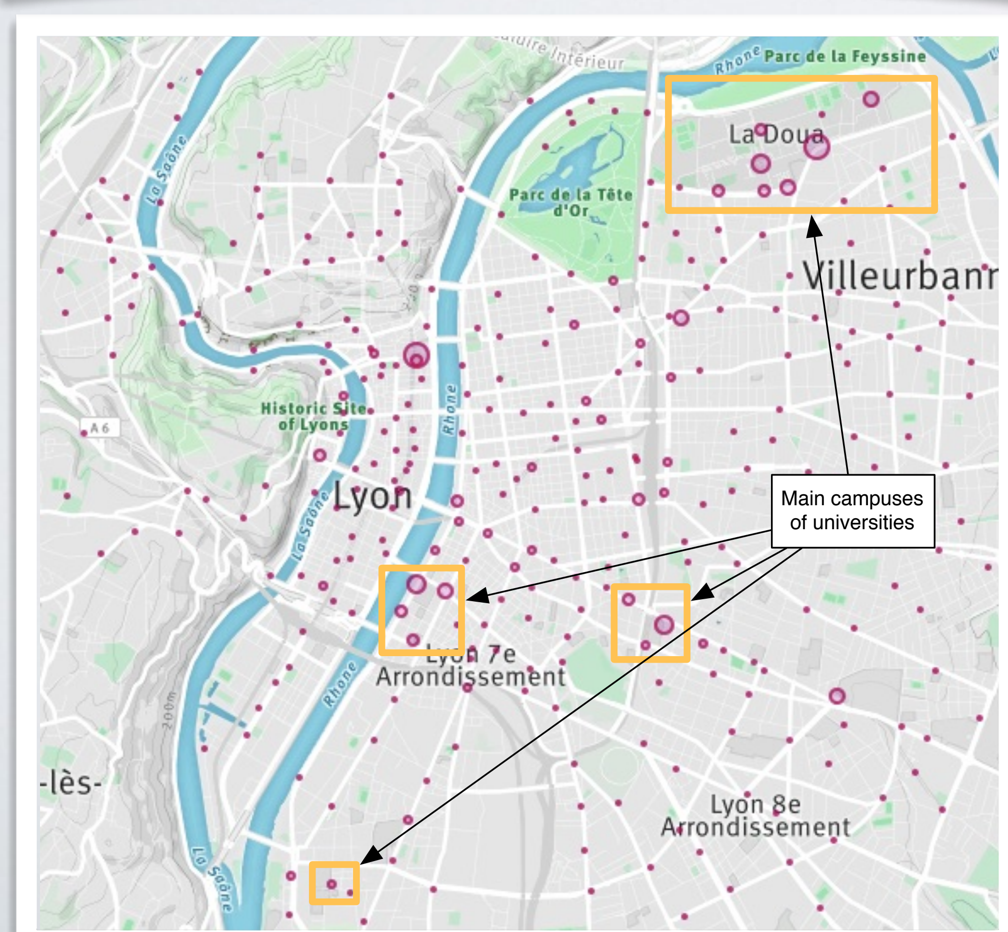


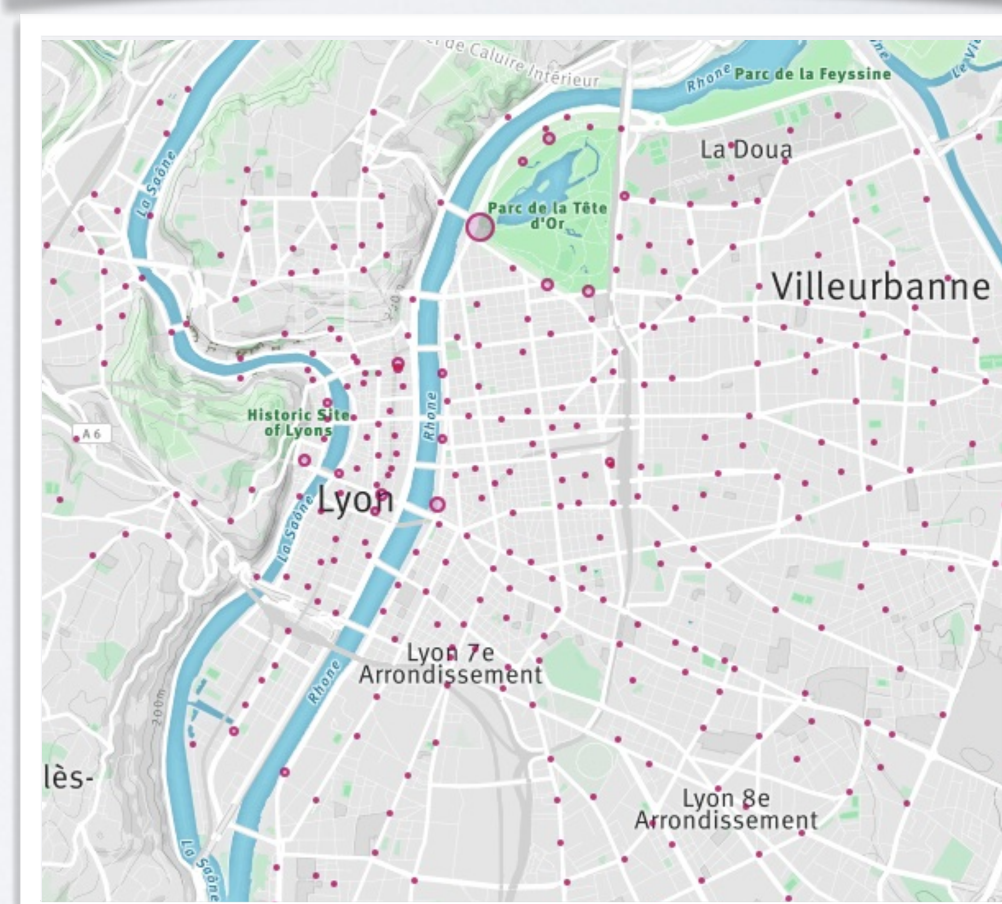
"Leisure"?




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


Main commercial street
Parc de la Feyssine
La Doua
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

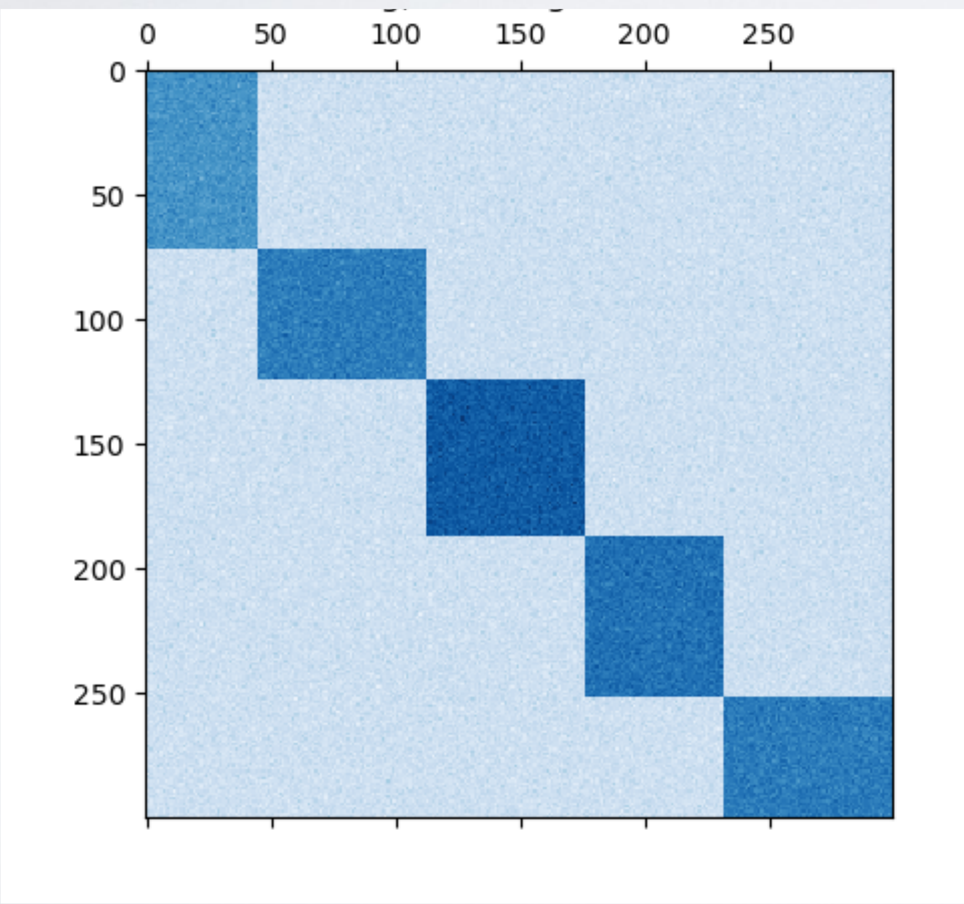

Parc de la Feyssine
La Doua
Parc de la Tête d'Or
Villeurbanne
Historic Site of Lyons
Lyon
Lyon 7e Arrondissement
Lyon 8e Arrondissement

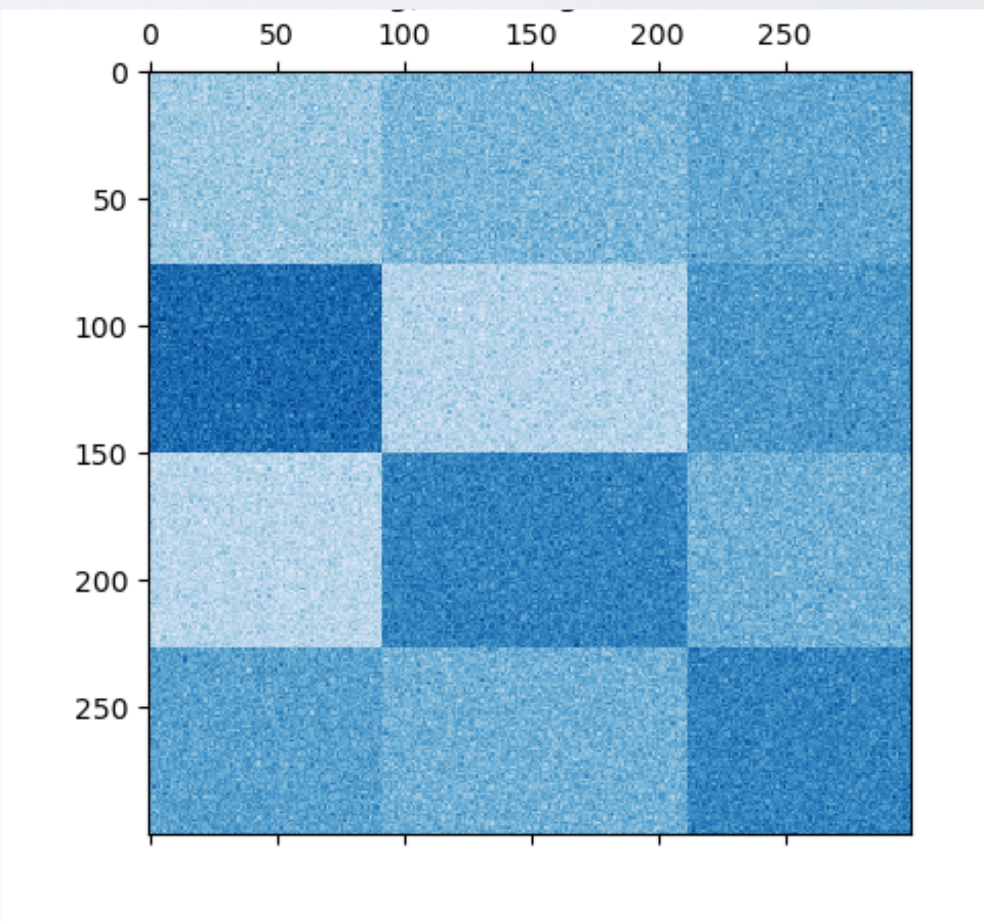
CO-CLUSTERING

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