Matrix Factorization
ALS: Alternating Least Squares

Nagiza F. Samatova, samatova@csc.ncsu.edu
Professor, Department of Computer Science
North Carolina State University

Senior Scientist, Computer Science & Mathematics Division
Oak Ridge National Laboratory

OAK RIDGE
National Laboratory

NC STATE UNIVERSITY
Department of Computer Science

Outline

- **Math Preliminaries**
  - **Graph Theory**
  - **Bipartite Graphs**
- **Math Preliminaries**
  - **Matrix Inverse, Transpose, Singular Matrix**
- **Matrix Factorization**
  - **SVD, PCA**
- **Regularization**
- **Alternating Least Squares for Matrix Completion**
  - **ALS Algorithm**
  - **ALS vs. SGD**
  - **ALS vs. SVD**
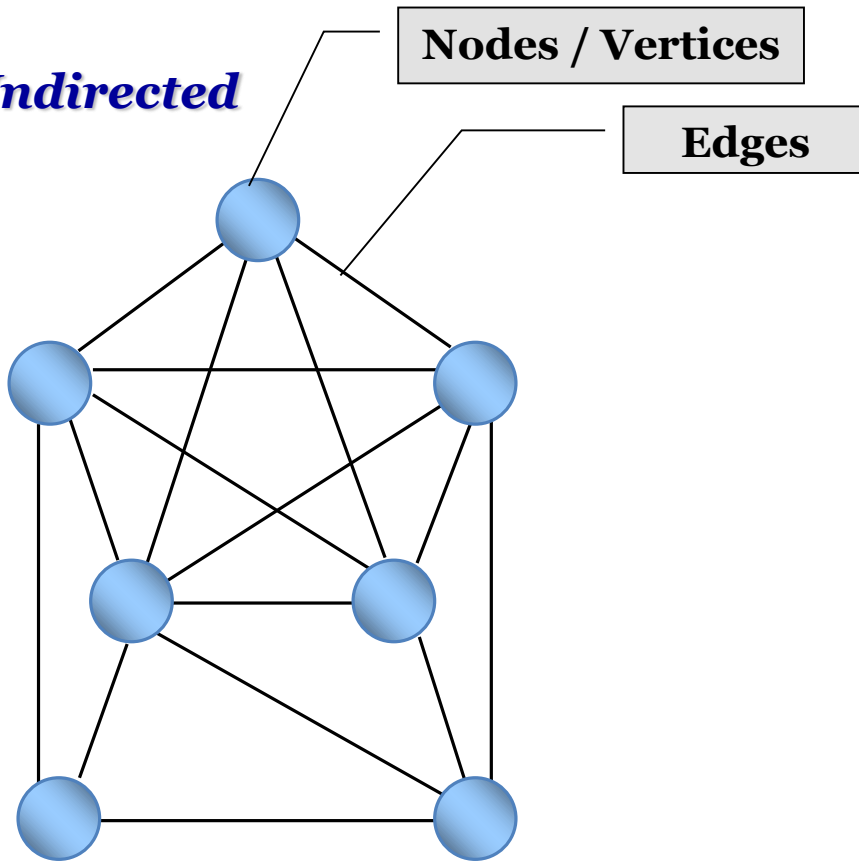- **Business Intelligence Use Case: Recommendation Systems**

Mathematical Preliminaries
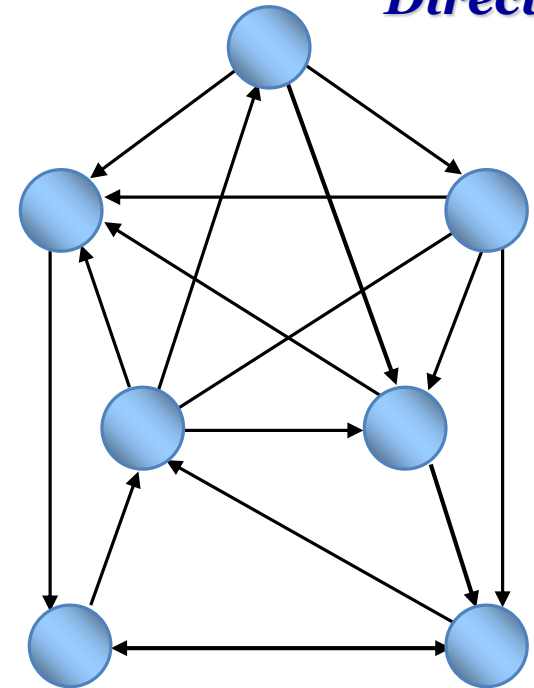## GRAPH THEORY

Graphs

Graph with 7 nodes and 16 edges

$$G = (V, E)$$
$$V = \{v_1, v_2, ..., v_n\}$$
$$E = \{e_k = (v_i, v_j) \mid v_i, v_j \in V, k = 1, ..., m\}$$

Nodes / Vertices

Edges

Undirected

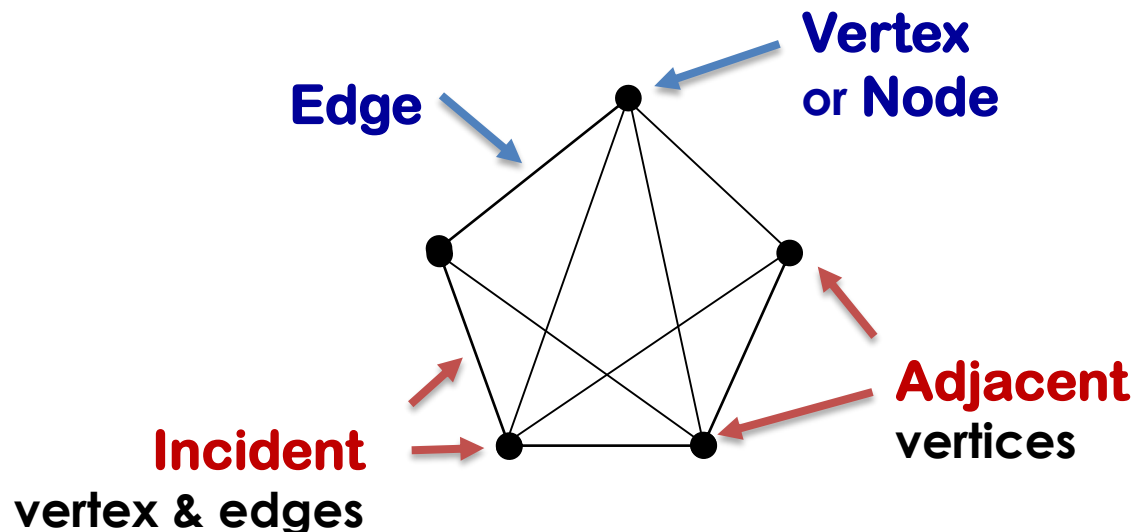*Directed*

$$(v_i, v_j) = (v_j, v_i)$$

$$(v_i, v_j) \neq (v_j, v_i)$$

Graph Theory

A **graph** is a collection of *vertices* (nodes) and *edges*.

A **vertex** represents some object.

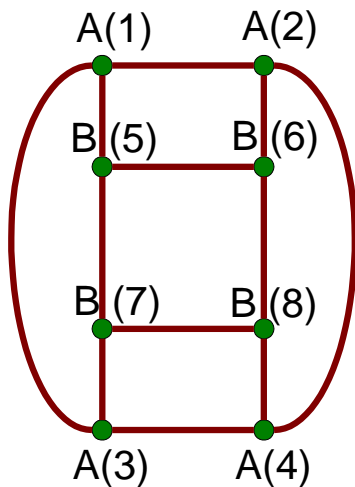An **edge** connects two vertices and represents some **relationship** between them.
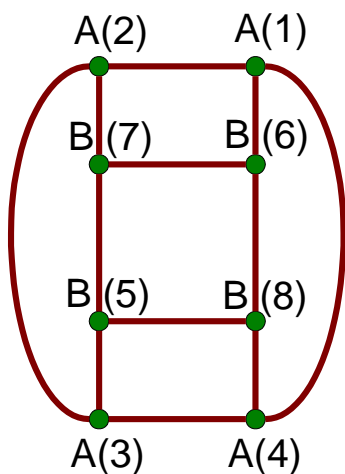
Graph Representation as a Matrix

- **Adjacency Matrix** (vertex vs. vertex)
- **Incidence Matrix (vertex vs. edge)**
- **Sparse** (lots of zero's) vs. **Dense** Matrices

Adjacency Matrix Representation

The representation is *NOT unique*.
Some algorithms are *order-sensitive*.

	A(1)	A(2)	A(3)	A(4)	B(5)	B(6)	B(7)	B(8)
A(1)	1	1	1	0	1	0	0	0
A(2)	1	1	0	1	0	1	0	0
A(3)	1	0	1	1	0	0	1	0
A(4)	0	1	1	1	0	0	0	1
B(5)	1	0	0	0	1	1	1	0
B(6)	0	1	0	0	1	1	0	1
B(7)	0	0	1	0	1	0	1	1
B(8)	0	0	0	1	0	1	1	1

	A(1)	A(2)	A(3)	A(4)	B(5)	B(6)	B(7)	B(8)
A(1)	1	1	0	1	0	1	0	0
A(2)	1	1	1	0	0	0	1	0
A(3)	0	1	1	1	1	0	0	0
A(4)	1	0	1	1	0	0	0	1
B(5)	0	0	1	0	1	0	1	1
B(6)	1	0	0	0	0	1	1	1
B(7)	0	1	0	0	1	1	1	0
B(8)	0	0	0	1	1	1	0	1

What apps naturally deal w/ graphs?

Social Networks

Semantic Web
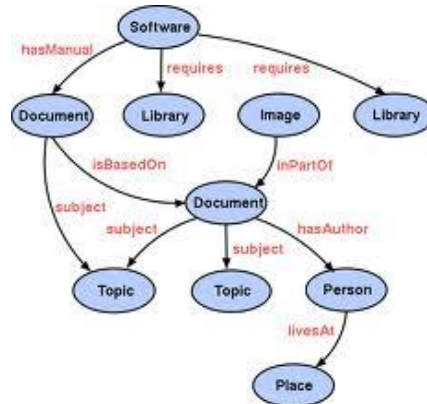
World Wide Web

Drug Design, Chemical compounds
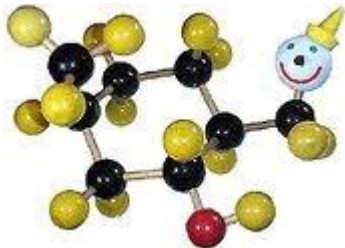
Computer networks

Sensor networks

Credit: Images are from Google images via search of keywords
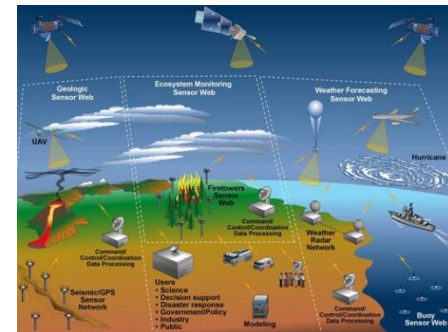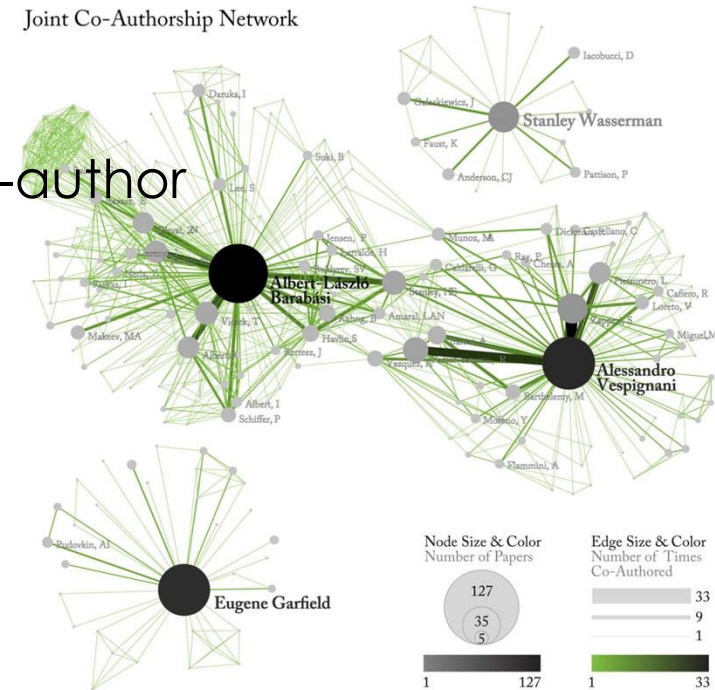
Real-world Graphs, or Networks

- **Graphs:** consist of objects (vertices) & their relationships (edges/links)
  - Social: Facebook, Twitter, Linkedin
  - Physical: Internet, power-grid
  - Biological: Protein-protein interactions
  - Academic Collaboration: Citation, co-author

Can you think of real-world graphs/networks?
- What are their nodes/vertices?
- What are their relationships/edges/links?

Joint Co-Authorship Network

Node Size & Color
Number of Papers

Edge Size & Color
Number of Times
Co-Authored

Examples of Real-World Graphs

Graph / Network	Node / Vertex	Edge / Arc / Relationship
Internet	Computer/router	Cable/wireless data connection
WWW	Web page	Hyperlink
Facebook	Person	Friendship
Power Grid	Generating station	Transmission line
Airport Network	Airport in a city	Flight connection
US Roads	City	Roads
Neural Network	Neuron	Synapse

How Big Are These Graphs?

Graphs encode rich relationships

# of Edges (y-axis): 58 million, 5.6 billion, 31 billion, 104 billion, 1 trillion, 8 trillion

# of Nodes (x-axis): 24 million, 504 million, 800 million, 1.4 billion, 50 billion, 1 trillion

- US Road Map
- Linked Data
- Facebook
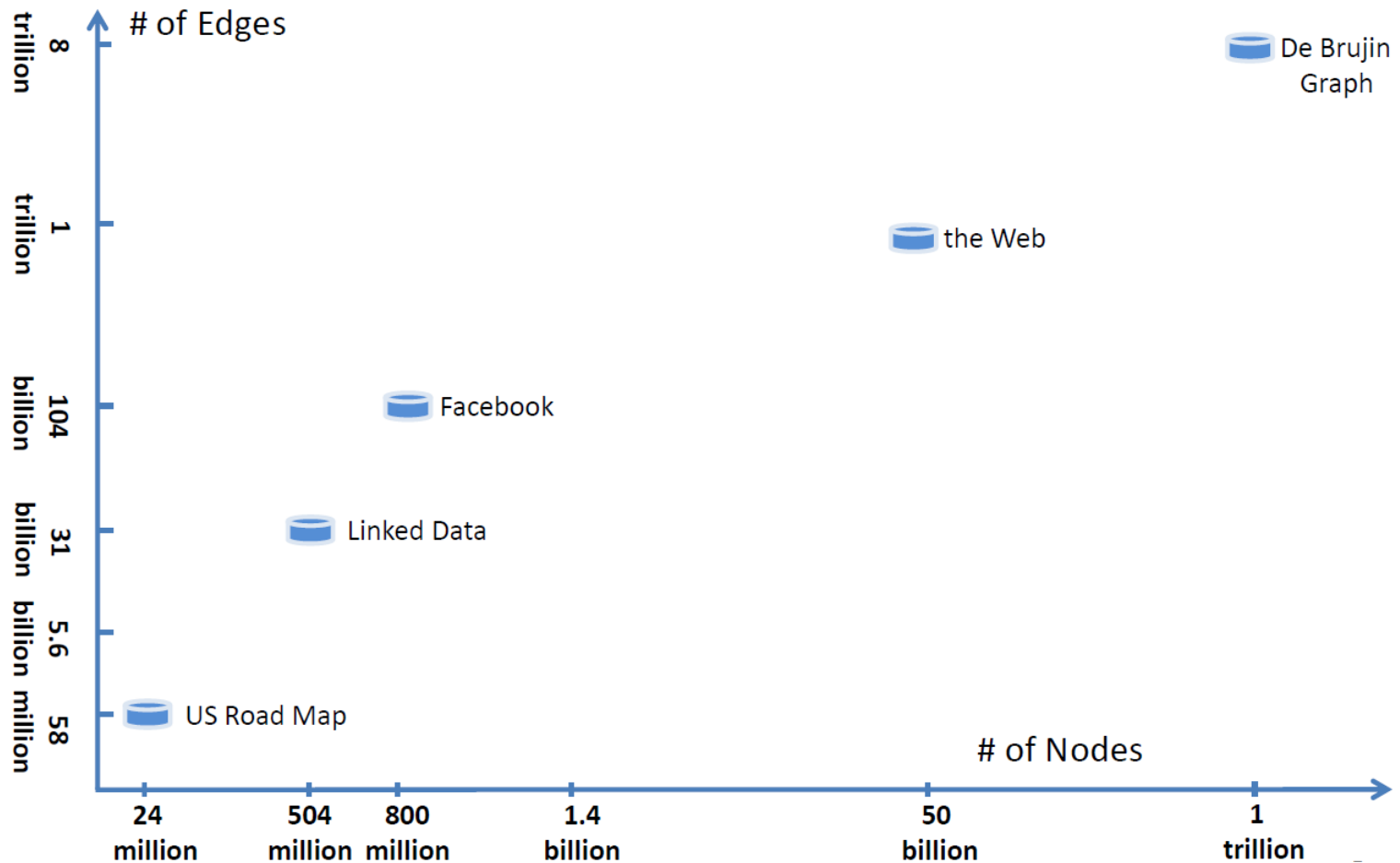- the Web
- De Brujin Graph

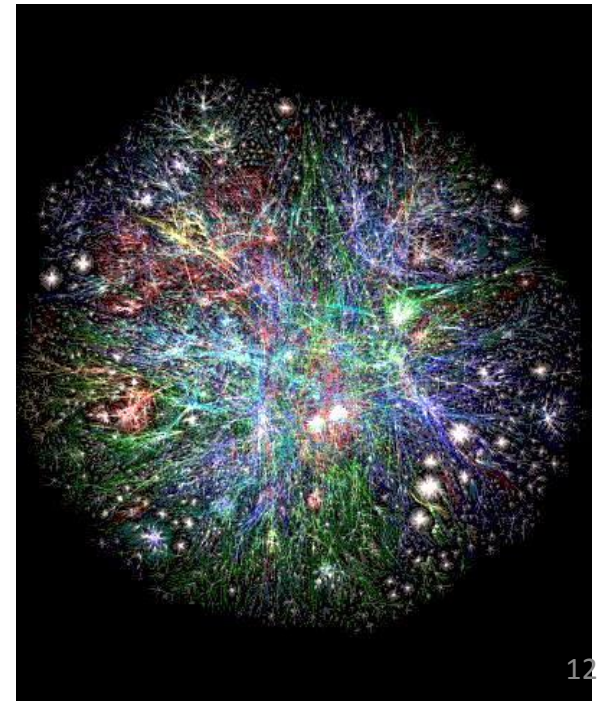Image source: Haixun Wang, KDD 2012

Graphs Challenge Many Algorithms

Massive scale

- – Can't use conventional algorithms on large graphs
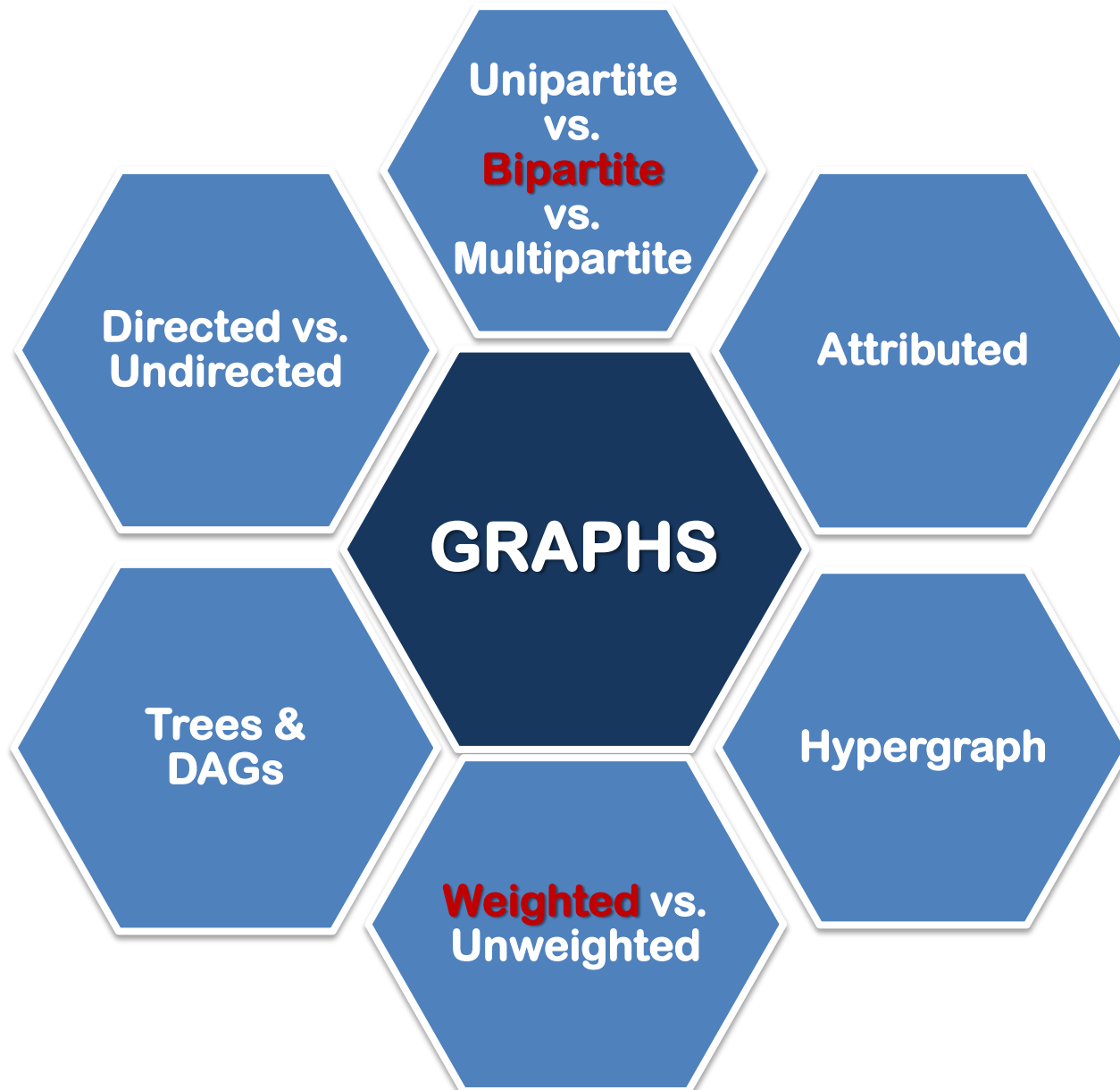- – The graph itself may not even fit in memory

Facebook: |V| = 721M, |E| = 137B
Common crawl: |V| = 3.5B, |E| = 128B

$$O(n^2) \;\text{☹}$$

Types of Graphs

Bipartite Graphs

A graph B is **bipartite** if its vertices V can be partitioned into two non-intersecting sets C and S such that all relationships/edges go between C and S (no edges go from C to C or from S to S.

$B = (V, R)$ is **bipartite** graph such that:
(1) $V = C \cup S$ and $C \cap S = \emptyset$
(2) $R = \{r_{cs} = (c, s): c \in C$ and $s \in S \}$
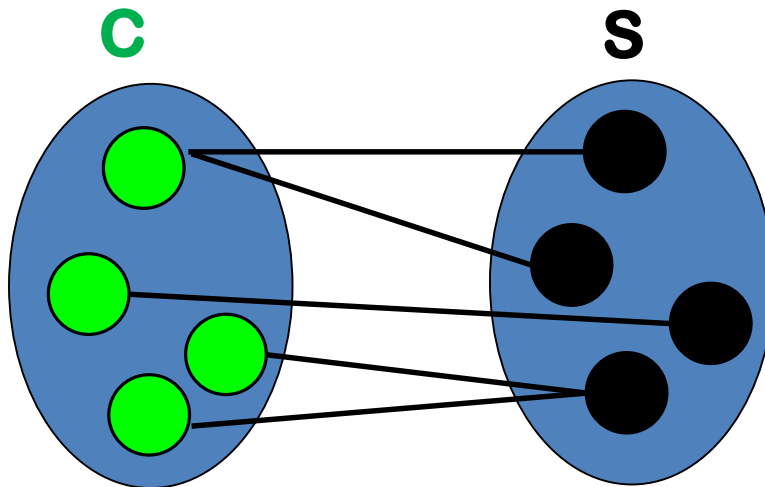
C   **S**   **Relation/Edge Types**

NO

YES

NO

Bipartite Graphs in Business Intelligence (BI)

$B = (V, R)$ is **bipartite** graph such that:
(1) $V = C \cup S$ and $C \cap S = \emptyset$
(2) $R = \{r_{cs} = (c, s): c \in C \text{ and } s \in S\}$
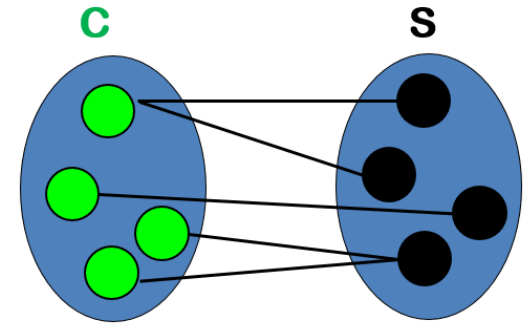
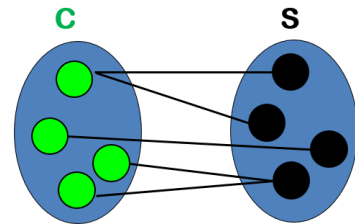| Business, B	Customers, C	Services, S	Relationships, R
Amazon	Customers	Products	Purchasing
Netflix	Subscribers	Movies	Watching
Pandora	Subscribers	Songs	Listening to
CISCO	Manufacturers	Parts	Supplying to CISCO

Observations about Bipartite BI Graphs

- Adjacency matrix is very **sparse**:($r_{cs} = 0$ or $r_{cs} =???$)
- Relationships are not always binary, but **weighted**:
  - $r_{cs} = *****$: How customer rated a product
  - $r_{cs} = 25$: How many times subscriber listened to the song
  - $r_{cs} = High$: How reliable manufacturer was in supplying the part

- **Missing** relationships ($r_{cs} = 0$ or $r_{cs} =???$) are **business opportunities**:
  - Recommend new products to customers
  - Find reliable manufacturers
  - Ensure continued customer subscriptions to services

Business, **B**	Customers, **C**	Services, **S**	Relationships, **R**
Amazon	Customers	Products	Purchasing /Rating
Netflix	Subscribers	Movies	Watching
Pandora	Subscribers	Songs	Listening to
CISCO	Manufacturers	Parts	Supplying to CISCO

Assumptions: Hidden Factors & Missing Links

There are **<u>a few</u> hidden** or **latent factors** $F = \{f_1, f_2, \ldots, f_h\}$ that define/drive the **relationships** between C and S.

These **latent / hidden factors** can be used **to complete missing relationships** between C and S.

Latent Factors, F

Predicted Links

Customers, C

Services, S

Observed Relationships, R
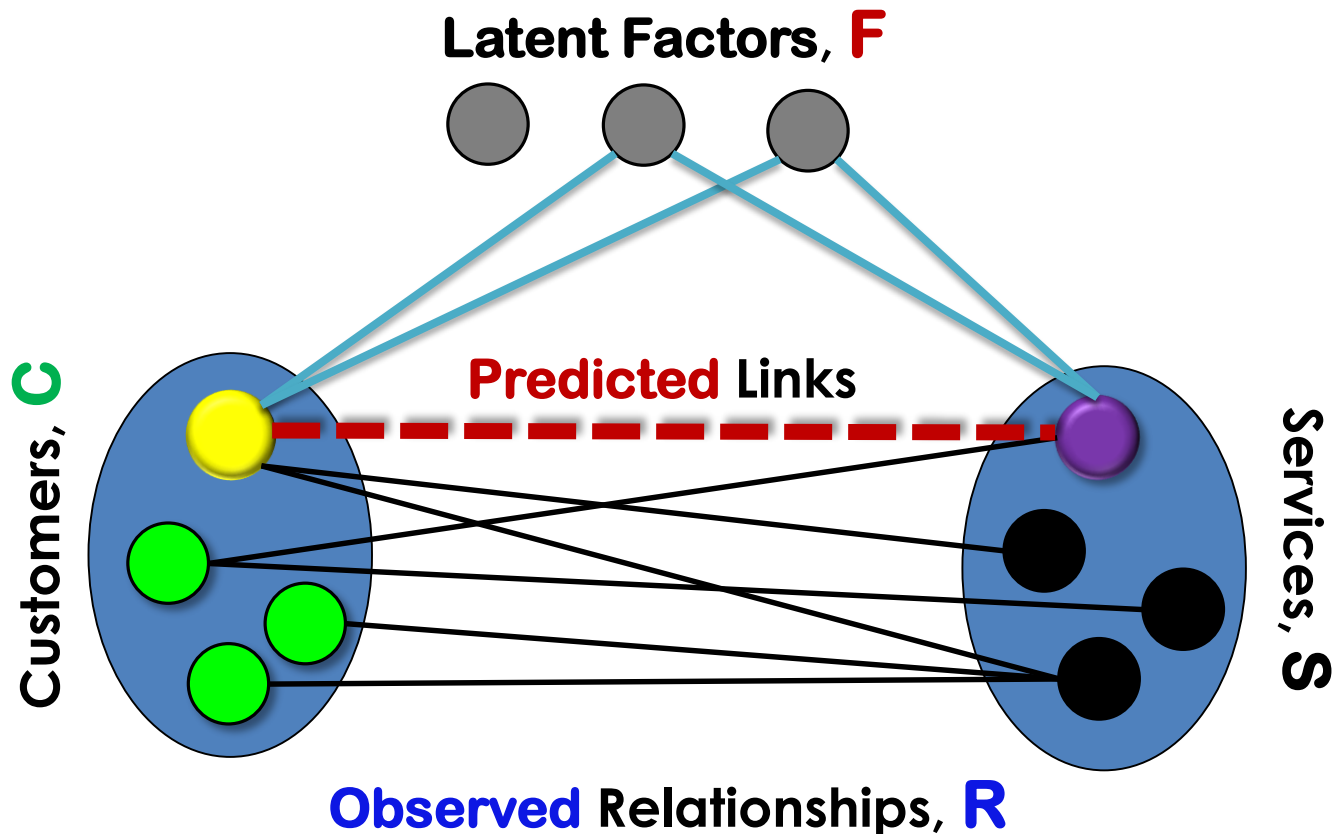
Example: Hidden Factors & Missing Links

Latent Factors, F:
- Tastes
- Genres

Pandora Customers, C

Recommend Song

Songs, S

R: Listened Song
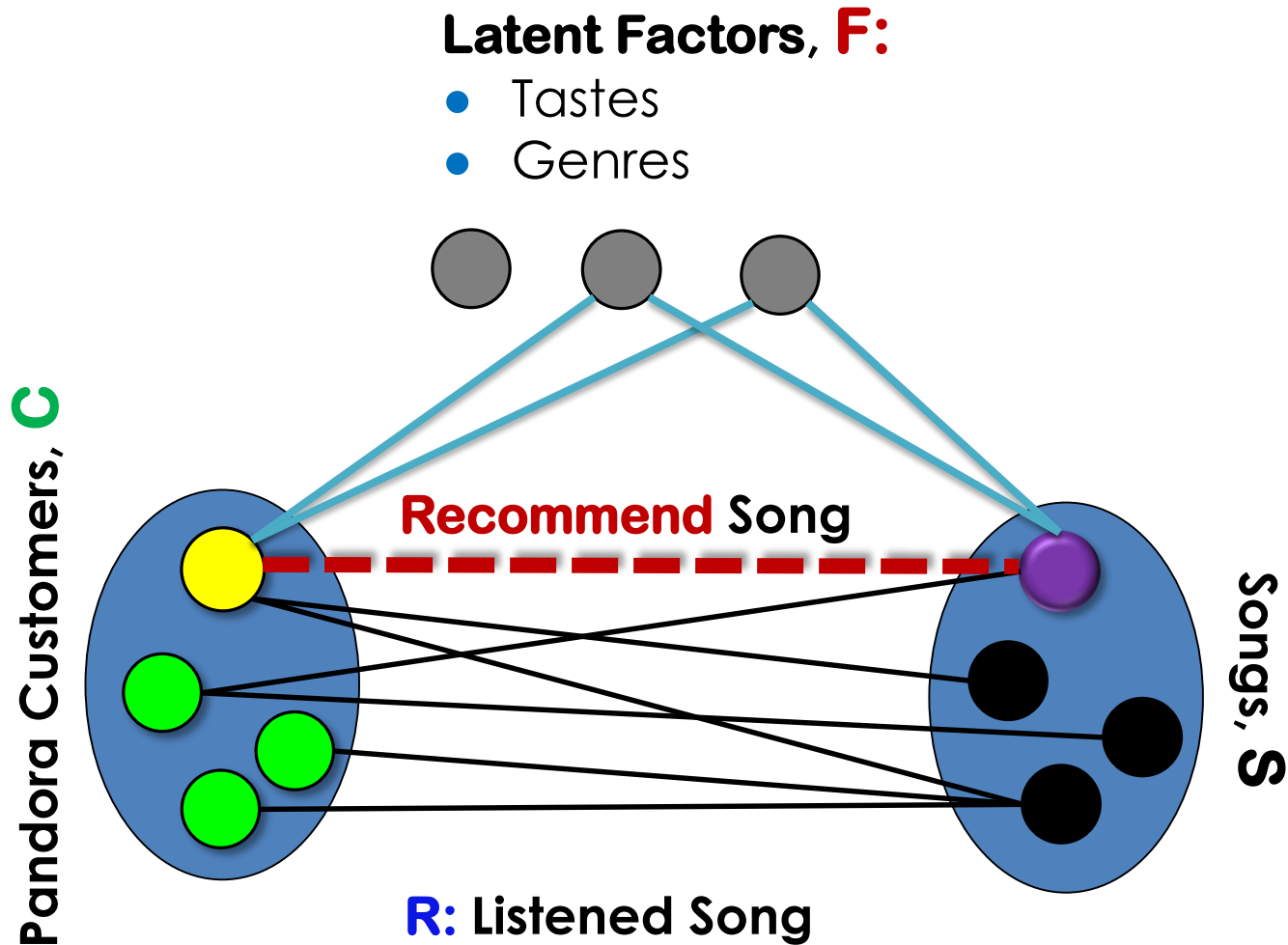
Latent Factor Space View

Customer-Service Relations in the **Latent Factor Space:**
$$F = \{f_1, f_2, \ldots, f_h\}$$

f_2

"Nearest-Neighbors" ➔
Complete Missing Link

f_1

f_3
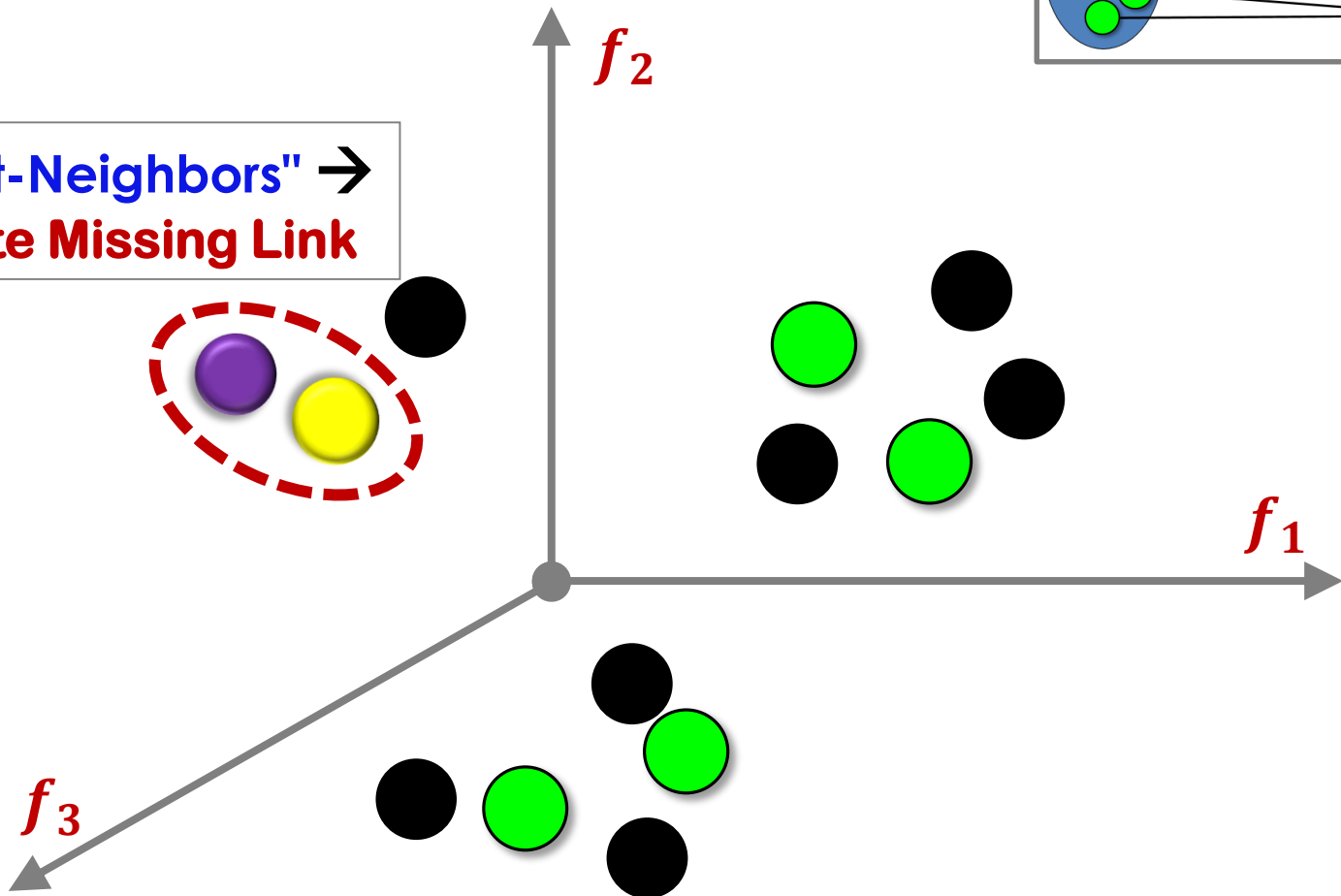
Ex: User-Movie Relations in a Latent Factor Space
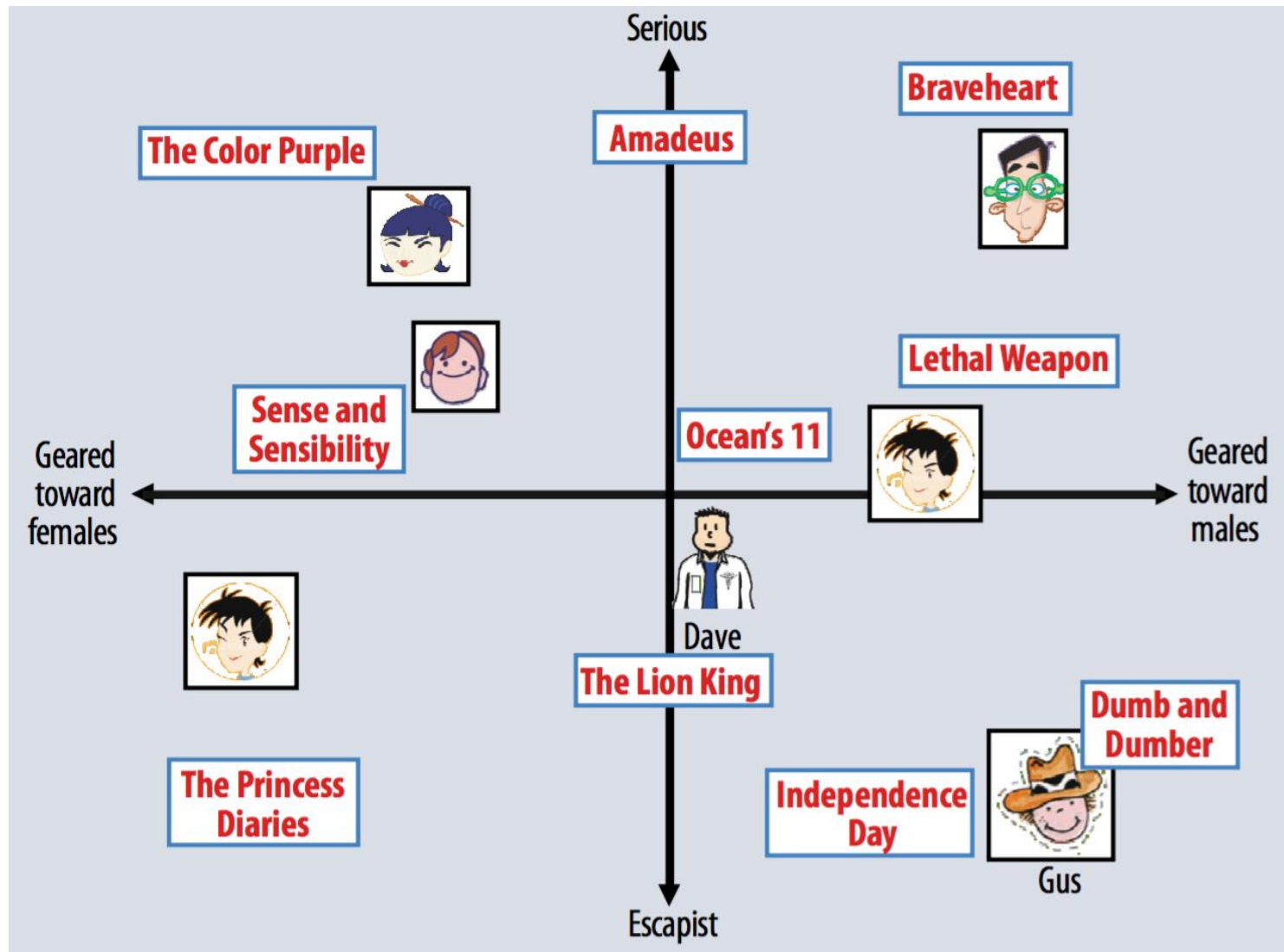
Matrix Factorization / Completion View

Services, S

Customers, C

5

?

Observed & Missing
Customer-Service
Relationships, **R**

Factors, **F**

Customer-**Factor**,
Relationship, **C**

Services, S

X

Transpose
Service-**Factor**,
Relationship, S^T
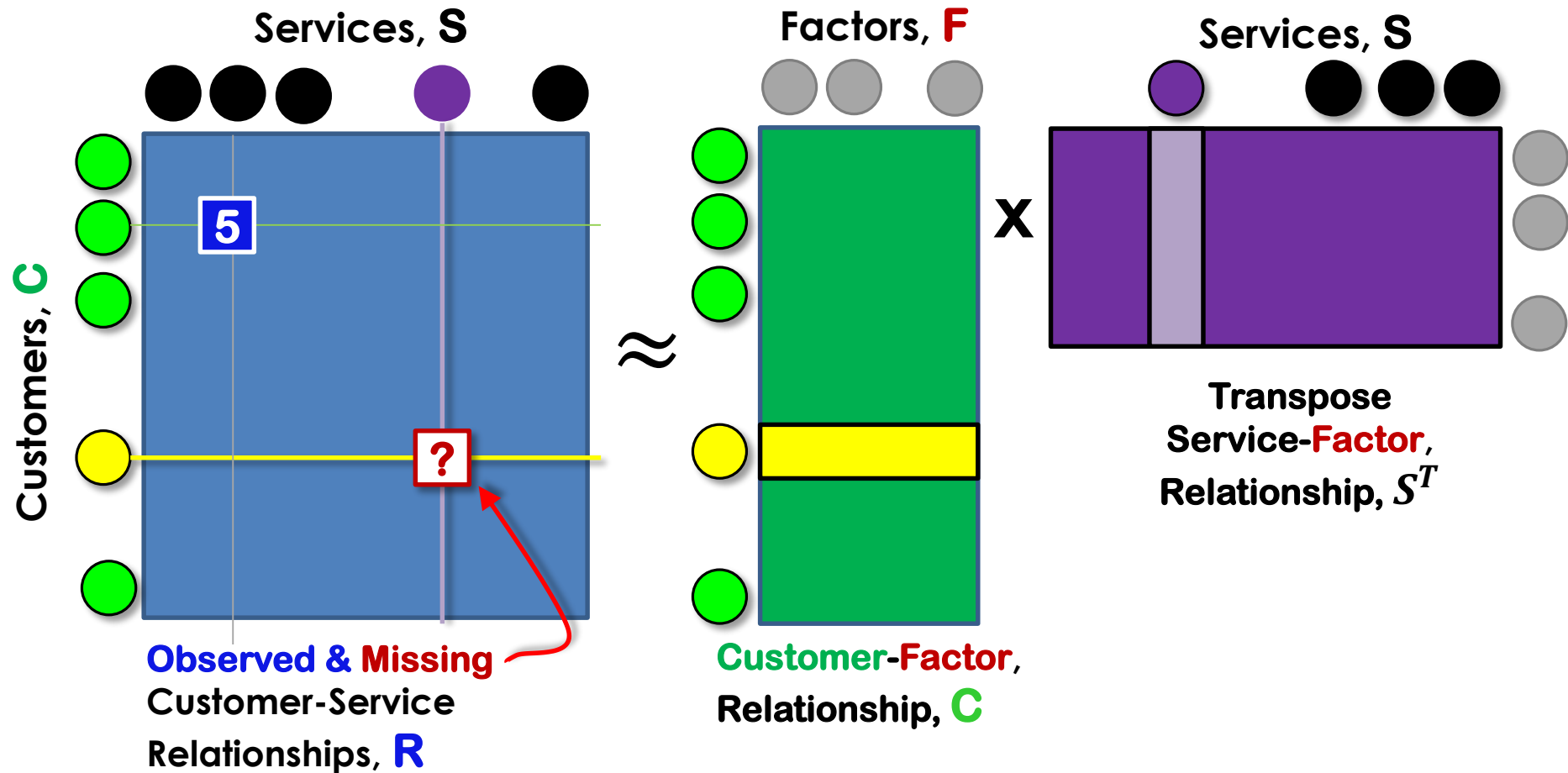
$$R \approx C \times S^T$$

Complete Missing Values

$$? \approx \quad \times$$

First Two Vectors from Matrix Decomposition
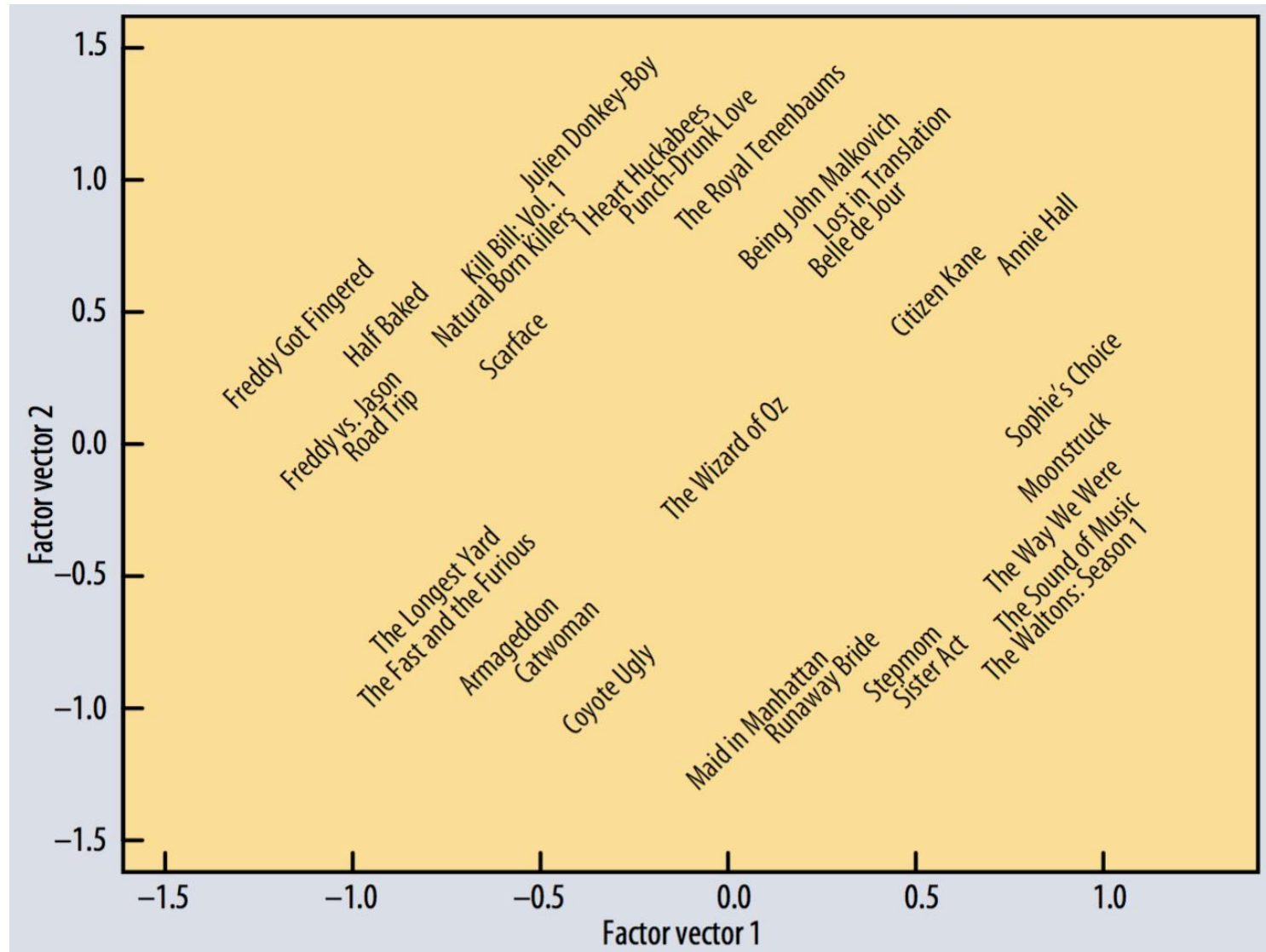
Matrix Factorization
ALTERNATE LEAST SQUARES

Paatero, 1994

Matrix Completion, or Matrix Factorization
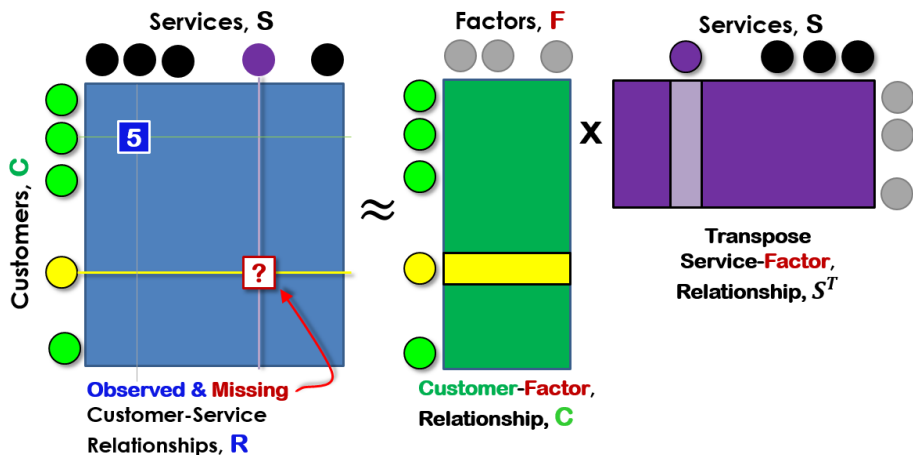
Customers: $\mathbf{C} = \{c_1, c_2, \ldots, c_n\}$

Services: $\mathbf{S} = \{s_1, s_2, \ldots, s_m\}$

Hidden Factors: $\mathbf{F} = \{f_1, f_2, \ldots, f_h\}$

$$R_{n \times m} \approx \widehat{R} = C_{n \times h} \times S^T_{h \times m}$$

$$r_{i,j} \approx \vec{c_i} \times \vec{s_j^T}$$

vector product

R: **sparse** matrix

$\widehat{R}$: **dense** matrix

$C \times S^T$: **dense** matrix

C: **dense** matrix

S: **dense** matrix

$h = O(constant)$

$h \ll n$

$h \ll m$

How to Find the "Best" $\widehat{R}$?

$$\boxed{R_{n \times m} \approx \widehat{R} = C_{n \times h} \times S^T_{h \times m}}$$

$$\boxed{r_{i,j} \approx \overrightarrow{c_i} \times \overrightarrow{s_j^T}}$$

known   unknown   unknown   vector product

- **For the sake of simplicity, <u>let's assume</u> that $S^T_{h \times m}$ is known.**
- **How to find $C_{n \times h}$?**
- **Can we multiply on the right by the inverse of $\left(S^T_{h \times m}\right)^{-1}$?**
  - Note: Matrix inverse is only for squared matrices
  - But $S^T_{h \times m}$ is skinny, rectangular ($h \ll m$)
- **How can we create a squared matrix?**

$$\boxed{R_{n \times m} \times S_{m \times h} \approx C_{n \times h} \times (S^T_{h \times m} \times S_{m \times h})}$$

- $S^T \times S$ **is an $h \times h$ squared matrix.**
- **<u>Let's assume</u> that $S^T \times S$ is invertible, then**

$$\boxed{R \times S \times \left(S^T \times S\right)^{-1} \approx C}$$

Assuming S is known and $S^T S$ is invertible

- <u>Goal</u>: Optimize the "best" approximation to the matrix **C**

$$R \times S \times \left(S^T \times S\right)^{-1} \approx C$$

Minimize Least Squares, or the Frobenius Norm

$$\left\| R \times S \times \left(S^T \times S\right)^{-1} - C \right\|_F \rightarrow min$$

$$E = R \times S \times \left(S^T \times S\right)^{-1} - C \qquad \text{Error Matrix, E}$$

$$\|E\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{h} E_{ij}^2} = \sqrt{tr(E^T E)}$$

Let's assume that $C_{n \times h}$ is known

$$R_{n \times m} \approx \widehat{R} = C_{n \times h} \times S_{h \times m}^T$$

$$r_{i,j} \approx \vec{c_i} \times \vec{s_j^T}$$

known   known   unknown   vector product

1. How to find $S_{h \times m}^T$?

2. What other assumptions should be made?

3. What is the optimization function?

$$??? \approx S$$

$$??? \rightarrow \mathbf{min}$$

Alternating Least Square

Optimization Problem: Find

$$R_{n \times m} \approx \widehat{R} = C_{n \times h} \times S^T_{h \times m}$$

Minimize Loss Function:

$$\left\| R - C \times S^T \right\|^2_F \to min$$

or equivalent

$$min_{C,S} \sum_{u,i} (r_{ui} - C_u S^T_i)^2 \qquad ★★★$$

- Optimizing **C** and **S** simultaneously is non-convex, hard
- If **C** <u>or</u> **S** are **fixed**, then the system of linear equations: **convex** & **easy**
  - Initialize **S** with random values
  - Solve for **C**
  - Fix **C**
  - Solve for **S**
  - **Repeat** ("Alternating")
    - Fixed number of iterations or
    - Till the Error stabilizes

S^T_i

Services, S

Factors, F

Services, S

r_{ui}

Customers, C

Transpose Service-**Factor**, Relationship, S^T

Observed & Missing
Customer-Service Relationships, **R**

Customer-**Factor**, Relationship, **C**

C_u
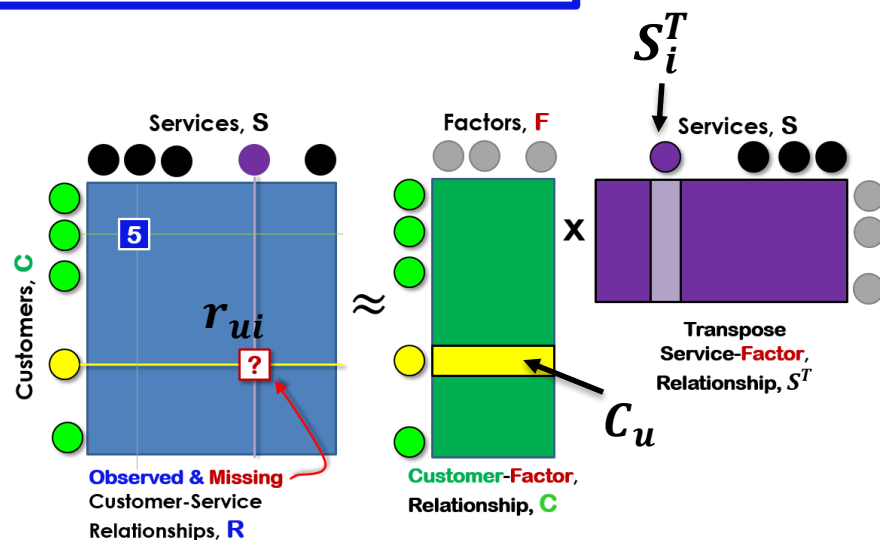
Practical Implementation

- With the ALS, the factorization just involves alternating solving for C and S by fixing the other.
    - When fixed, it is a problem that has a **direct analytical solution**, one which is **entirely and trivially parallelizable (by row)**.
    - In practice, the **inverses are never computed**:
        - Use the **QR decomposition** because it's **fast** and can 'detect' when the requested rank is even too low.

Machine Learning Background
REGULARIZATION

Regularization is a technique to solve the overfitting problem.

"Regularization artificially discourages complex or extreme explanations of the world even if they fit what has been observed better. The idea is that such explanations are unlikely to generalize well to the future; they may happen to explain a few data points from the past well, but this may just be because of accidents of the sample."

Sean Owen, Director, Data Science @ Cloudera

Overfitting Problem: Motivational Example

- First of all, I want to clarify how this problem of overfitting arises.
- Let's model a problem, to predict the wage based age.
- A linear regression model with age as an independent variable and wage as a dependent one. This model will mostly fail, since it is too simple.
- Let's add the sex and the education as explaining variables. The model becomes more interesting and more complex. You measure its accuracy regarding a loss metric $L(X,Y)$ where X is the design matrix and Y is the response vector (here the wages).You find out that your result are quite good but not as perfect as you wish.
- Add more variables: location, profession of parents, social background, number of children, weight, number of books, preferred color, best meal, last holidays destination... and so on.
  - Your model will do good but it is probably **overfitting**, i.e. it will probably have **poor prediction and generalization power**: it sticks too much to the data and we have probably learnt the background noise while fitting the model. This isn't acceptable.
- So how do you solve this? It is here the regularization technique comes handy. You **penalize your loss function by adding a multiple of an L1 (LASSO) or an L2 (Ridge) norm of your weights vector w** (it is the vector of the learned parameters in your linear regression).

L_1- and L_2- Norm of a Vector

- Let's consider:
  - a 2−dimensional vector: $\vec{w} = (w_1, w_2)$
  - a constant $\lambda \geq 0$
- **L_1-norm**, $\|\vec{w}\|_1 = |w_1| + |w_2|$
- **L_2-norm**, $\|\vec{w}\|_2 = w_1^2 + w_2^2$

$\vec{w} = (w_1, w_2, ..., w_d)$

$$\|\vec{w}\|_1 = |w_1| + |w_2| \leq \lambda$$
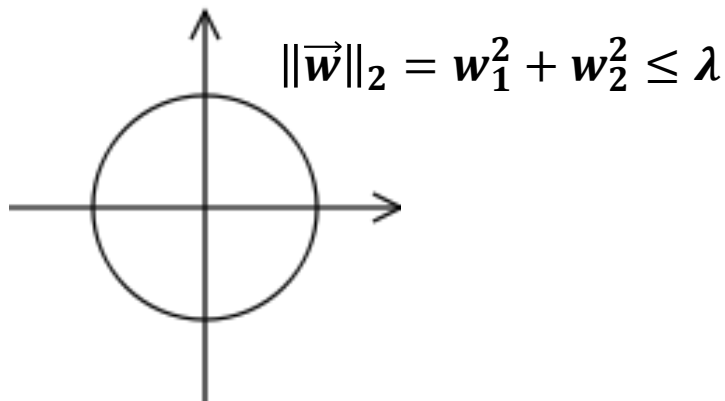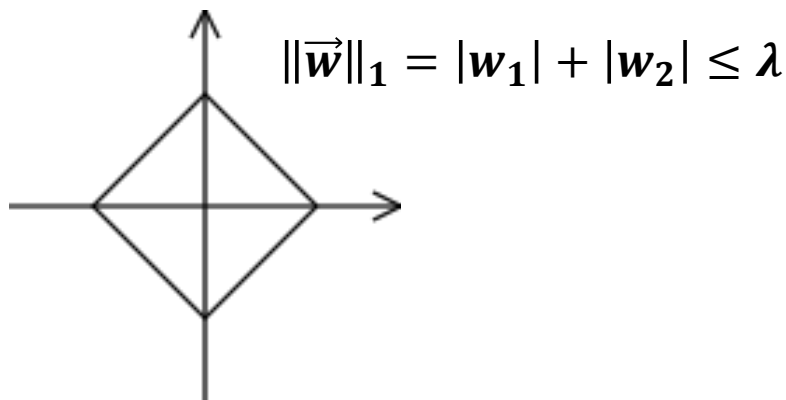
$$N(\vec{w}) = \|\vec{w}\|_1 = \sum_{i=1}^{d} |w_i|$$

$$N(\vec{w}) = \|\vec{w}\|_2 = \sqrt{\sum_{i=1}^{d} w_i^2}$$

$$\|\vec{w}\|_2 = w_1^2 + w_2^2 \leq \lambda$$

$$N(\vec{w}) \geq 0$$

Loss Function & Regularization/Shrinkage

Optimization Problem: **Find parameters** $\vec{w} = (w_1, w_2, ..., w_d)$ **:**

Minimize Non-negative Loss Function: $Loss(\vec{w}) \rightarrow min$

Constraint on the Norm: $0 \leq N(\vec{w}) \leq \lambda$

● **This Problem statement is <u>equivalent</u> to:**

Minimize Loss Function: $\boxed{Loss(\vec{w}) + \lambda \times N(\vec{w}) \rightarrow min}$

regularization term

● **Regularization, or Shrinkage:**
  ● Forces the parameters to stay within a certain manifold
    ● e.g., hypersphere, in case of L_2-norm
  ● **Shrinks the parameters**

Example: Linear Regression w/ Regularization

<u>Optimization Problem:</u> **Find parameters** $\vec{\beta} = (\beta_1, \beta_2, ..., \beta_d)$ **:**

Minimize Loss Function: $Loss(\beta_0, \vec{\beta}) = \sum_{i=1}^{n}(f(X_i, \beta_0, \vec{\beta}) - Y_i)^2 \rightarrow min$

$f() = \widehat{Y_i}$: linear function
of **parameters** $\beta's$

Constraint on the Norm: $N(\vec{\beta}) \leq \lambda$

- **This Problem statement is <u>equivalent</u> to:**

Minimize Loss Function: $Loss(\beta_0, \vec{\beta}) + \lambda \times N(\vec{\beta}) \rightarrow min$

Ridge Regression: $Loss(\beta_0, \vec{\beta}) + \lambda \times \left\|\vec{\beta}\right\|_1 \rightarrow min$

Lasso Regression: $Loss(\beta_0, \vec{\beta}) + \lambda \times \left\|\vec{\beta}\right\|_2 \rightarrow min$

L_1-regularization vs. L_2-regularization

$$N(\vec{\boldsymbol{w}}) = \|\vec{\boldsymbol{w}}\|_1 = \sum_{i=1}^{d} |w_i|$$

$$N(\vec{\boldsymbol{w}}) = \|\vec{\boldsymbol{w}}\|_2 = \sqrt{\sum_{i=1}^{d} w_i^2}$$

L_1-regularization	L_2-regularization
Computationally inefficient on non-sparse cases	Computationally efficient due to having analytical solutions
Sparse outputs	Non-sparse outputs
Built-in feature selection	No feature selection

Regularization
ALTERNATE LEAST SQUARES

Matrix Factorization

- **Approach**
  - **Fix a relatively small number h (e.g., $h = 10$)**
  - **Summarize each customer u with an $h-$dimensional vector $C_u = (C_{u1}, C_{u2}, \ldots, C_{uh})$ and each service i with $h-$ dimensional vector $S_i = (S_{i1}, S_{i2}, \ldots, S_{ih})$.**
  - **The vectors C_u and S_i are referred to as factors**
  - **To predict customer u's rating for service i , calculate $r_{ui} \approx C_u^T S_i$**
  - **In the matrix form:**
    - Let $C_1, \ldots, C_n \in \mathbb{R}^h$ be the factors for the customers
    - and $S_1, \ldots, S_m \in \mathbb{R}^h$ be the factors for the services
    - The $h \times n$ customer matrix C, and the $h \times m$ service matrix S are defined by:

    - $$C = \begin{bmatrix} | & & | \\ C_1 & \ldots & C_n \\ | & & | \end{bmatrix}, \; S = \begin{bmatrix} | & & | \\ S_1 & \ldots & S_m \\ | & & | \end{bmatrix}$$

    - Our goal is to estimate the complete ratings matrix $R \approx C^T S$

Least Square Error

- **Objective: Minimize the least square error of the observed ratings**

$$\min_{C,S} \sum_{r_{ui}\,observed} \left(r_{ui} - C_u^T S_i\right)^2$$

with regularization term to avoid overfitting

$$\min_{C,S} \sum_{r_{ui}\,observed} \left(r_{ui} - C_u^T S_i\right)^2 + \lambda \left(\sum_u \|C_u\|^2 + \sum_i \|S_i\|^2 \right) \quad \$\$$$

λ is **regularization parameter**

- The constant λ: controls the extent of regularization, is usually determined by cross validation

This objective function is non-convex because of the $C_u^T S_i$ term

- It is NP-hard to optimize
- **Gradient descent** can be used as an approximation here, how it turns out to be slow and takes a lot of iterations to converge

Adding Biases (see paper for Paper Reflection)

- **Adding biases**
  - A rating is created by also adding biases
  $$\widehat{r_{ui}} = \mu + b_i + b_u + S_i^T C_u$$

- **Objective Function**
  - minimize the regularized squared error
  $$min_{b,S,C} \sum_{(u,i)} (r_{ui} - (\mu + b_i + b_u + S_i^T C_u))^2 +$$
  $$+\lambda(b_i^2 + b_u^2 + \|S_i\|^2 + \|C_u\|^2)$$

  - Minimization is typically performed by either stochastic gradient descent or alternating least squares

R vs $\widehat{R}$: $h = 3, \lambda = 2, \alpha = 40, 10$ iterations

Alternating Least Squares

- **Make objective convex**
  - **If we fix the set of variables C and treat them as constants, the objective is a convex function of S**
  - **If we fix the set of variables S and treat them as constants, the objective is a convex function of C**
- **Approach**
  - **Fix S and optimize C, then fix C and optimize S, and repeat until convergence.**

Alternating Least Squares Algorithm

1. Initialize C, S

2. repeat until convergence

3.   for $u = 1..n$

4. $C_u = \left(\sum_{r_{ui} \in r_{u*}} S_i S_i^T + \lambda I_h \right)^{-1} \sum_{r_{ui} \in r_{u*}} r_{ui} S_i$

5.   for $i = 1..m$

6. $S_i = \left(\sum_{r_{ui} \in r_{*i}} C_u C_u^T + \lambda I_h \right)^{-1} \sum_{r_{ui} \in r_{*u}} r_{ui} C_u$

ALS: Computational Cost

- **Computational cost**
  - Updating each $\boldsymbol{C_u}$ will cost $O(n_u h^2 + h^3)$ where n_u is the number of services observed for customer u , and similarly updating each $\boldsymbol{S_i}$ will cost $O(n_i h^2 + h^3)$ where n_i is the number of users that have rated service i
- **Cost to make prediction**
  - To simply predict $r_{ui} \approx C_u^T S_i$ for each customer u and service i will cost $O(nmh)$ if we estimate every user-service pair.
    - This is prohibitive for most datasets.
    - More feasible alternative is to use $\boldsymbol{C_u}$ and $\boldsymbol{S_i}$ as features in another learning algorithm

ALS vs. SVD

- Alternating Least Squares is flexible but less precise. It refers to any means of factoring $R \approx CS^T$, where C and S are low rank.
  - **Minimizes some squared-error difference** with the input R
  - The **loss function** can be **customized**:
    - Ignore missing values (crucial) or weight different r_{ui}'s differently.
    - The price: don't get many guarantees about the two factors:
      - They are not necessarily orthonormal.
  - Iterative: Ok to answer fast and refine later as needed
  - Faster than SVD but dumber than SVD

- Singular Value Decomposition (SVD) gives more guarantees about its factorization: $\boldsymbol{R = U \times \Sigma \times V^T}$
- The two outside factors (U and V) are orthornormal.
- $\boldsymbol{\Sigma}$ (eigenvalues/singular values) help determine how big ***h*** should be.
- SVD is relatively more **computationally expensive** and **harder to parallelize**.
- There is also not a good way to deal with **missing values** or **weighting**; you need to assume that in your sparse input, missing values are equal to a mean value 0.

ALS vs. SGD

- In an **SGD** (**Stochastic Gradient Sescent**) approach, for each example in the dataset you compute the error $r_{ui} - C_u S_i^T$ and then you update the parameters by a factor in the opposite direction of the gradient.
- In the **ALS** (**Alternating Least Squares**), you can **turn the non-convex optimization problem in Equation (***) into an "easy" quadratic problem** if you fix either C_u or S_i^T.
  - ALS fixes each one of those *alternatively*.
  - When one is fixed, the other one is computed, and vice versa.

- **Benefits of ALS over SGD**
  - ALS is very easy to parallelize and efficient.
  - SGD is not practical (users times items ~ billions).
  - In SGD, you are repeatedly picking some subset of the loss function to minimize -- one or more cells in the rating matrix.
  - In ALS you're **minimizing the entire loss function at once**, but, only twiddling half the parameters. That's because the optimization has an easy algebraic solution -- if half your parameters are fixed.
    - There is no gradient in the optimization step since each optimization problem is convex and doesn't need an approximate approach.
    - But, each problem you're solving is not the "real" optimization problem -- you fixed half the parameters.

Business Intelligence Use Case
RECOMMENDATION SYSTEMS

In our case [Netflix], more than 75% of what people choose come from recommendations.

Xavier Amatriain, former Director of Research @ Netflix

*A recommender system aims at providing a personalized list of items ranked according to the preferences of the user, as such **ranking methods are at the core of many recommendation algorithms***.

Problem: Recommender Systems (RS)

- **RS seen as a function**
- **Given:**
  - **User model** (e.g. ratings, preferences, demographics, situational context)
  - **Items** (with or without description of item characteristics)
- **Find:**
  - **Relevance score**. Used for ranking.
- **Finally:**
  - **Recommend relevant items**
- **But:**
  - Remember that relevance might be context-dependent
  - Characteristics of the list itself might be important (diversity)

Formally, recommendation problem

- **Problem**
  - **Given: the ratings (implicit / explicit) that users provided for certain items**
  - **Task: predict the ratings for the rest of the items**
- **Formally**
  - **If there are n users and m items, we are given an $n \times m$ matrix R in which the r_{ui} is the rating value provided by user u for an item i.**
  - **The goal is to estimate the unobserved (missing) entries in the matrix R**

RS: Applications

- **Online Shopping & Advertising**
  - Frequently co-purchased items are recommended
- **Entertainment**
  - Movie you may enjoy or Songs you may like
  - Cell phone games you may want to play
  - News you may want to read
- **Social Web**
  - Jobs you may be interested in
  - Groups you may like
  - Experts whose opinion might be relevant
  - Friends you may connect to
- **Finance & Accounting**
  - Stocks that may be in trouble
- **Analytic Workflows**
  - Personal assistance
  - What application feature to look at

RS: Business Intelligence
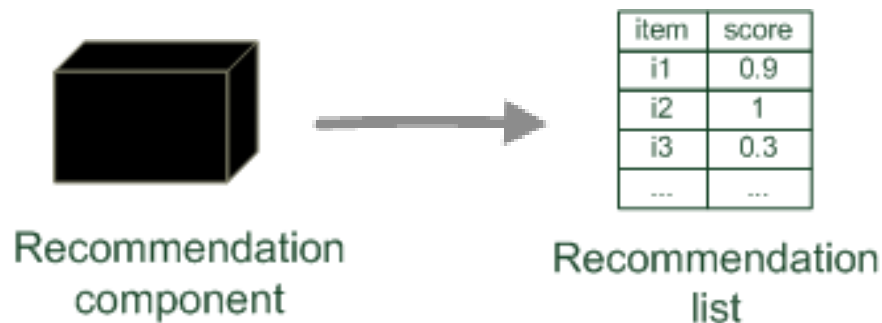
- **Value for the customer**
  - Find things that are interesting
  - Narrow down the set of choices
  - Help explore the space of options
  - Discover new things
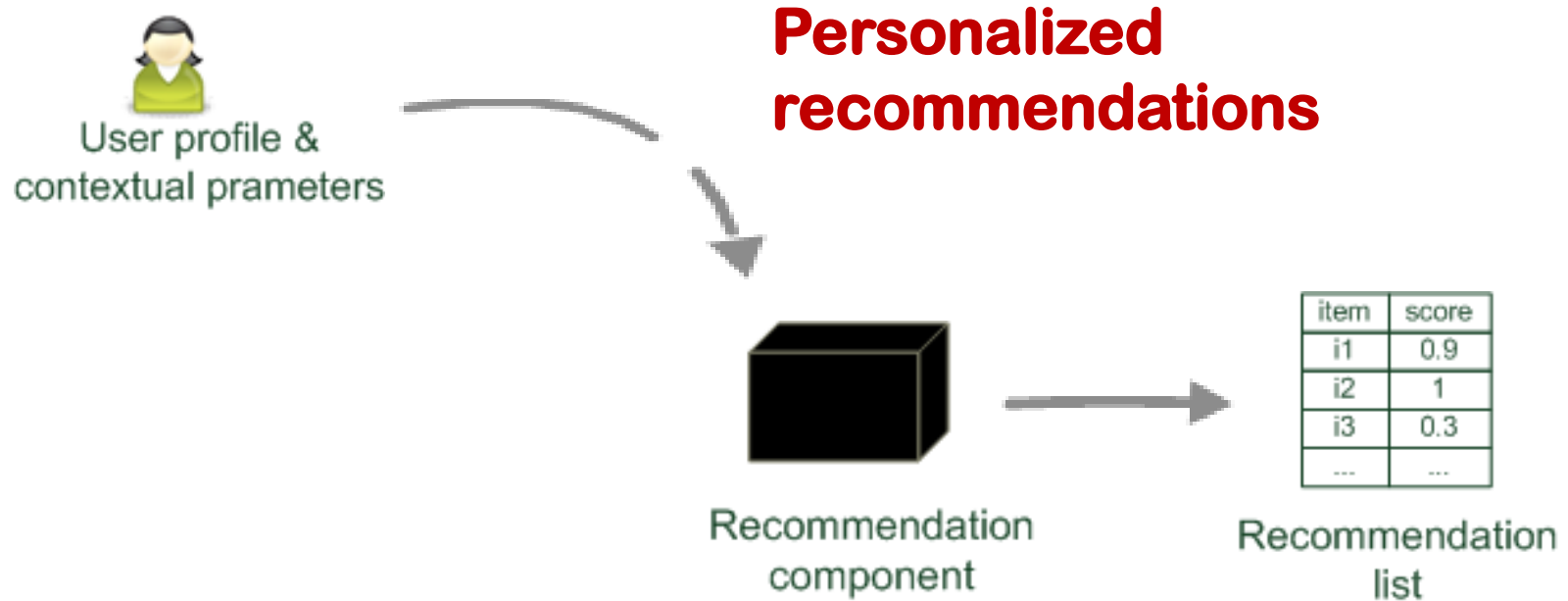 - Entertainment
  - …
- **Value for the provider**
  - Personalized service for the customer
  - Increase trust and customer loyalty
  - Increase sales, click trough rates, conversion etc.
  - Opportunities for promotion, persuasion
  - Obtain more knowledge about customers
  - …

Paradigms of Recommender Systems

RS reduce information overload
by estimating relevance

Recommendation
component

item	score
i1	0.9
i2	1
i3	0.3
...	...

Recommendation
list

Paradigms of Recommender Systems

User profile &
contextual prameters

**Personalized
recommendations**

Recommendation
component

item	score
i1	0.9
i2	1
i3	0.3
...	...

Recommendation
list

Paradigms of recommender systems

Collaborative: "Tell me what's popular among my peers"

User profile & contextual prameters

Community data

Recommendation component

item	score
i1	0.9
i2	1
i3	0.3
...	...

Recommendation list

Paradigms of Recommender Systems

Content-based: "Show me more of the same what I've liked"

Paradigms of Recommender Systems

Knowledge-based: "Tell me what fits based on my needs"

User profile & contextual prameters

Title | Genre | Actors | ...

Product features

Knowledge models

Recommendation component

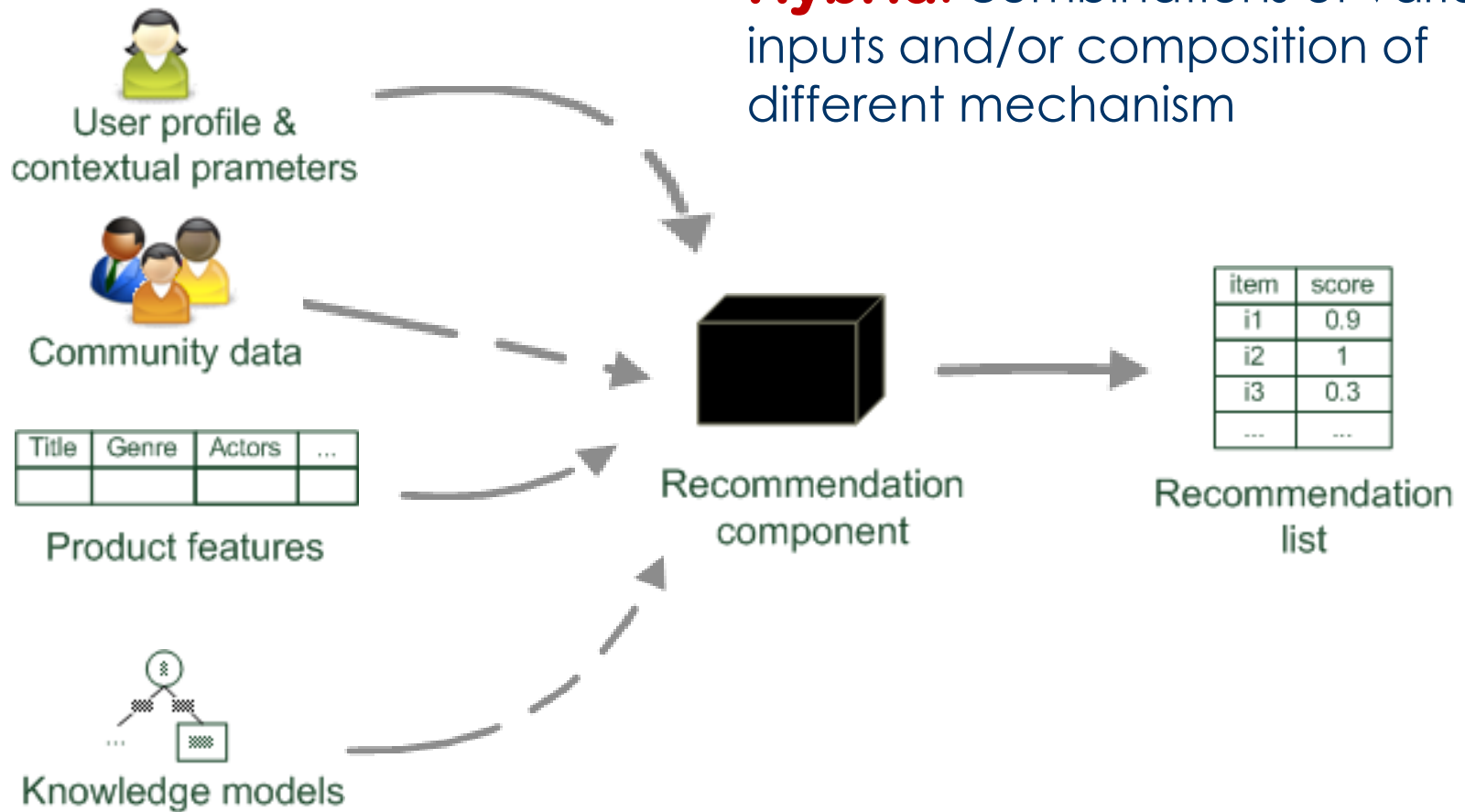| item	score
i1	0.9
i2	1
i3	0.3
...	...

Recommendation list

Paradigms of Recommender Systems

Hybrid: combinations of various inputs and/or composition of different mechanism

User profile & contextual prameters

Community data

Title	Genre	Actors	...

Product features

Knowledge models

Recommendation component

item	score
i1	0.9
i2	1
i3	0.3
...	...

Recommendation list

# Recommender Systems: Basic Techniques

	Pros 👍	Cons 👎
Collaborative	No knowledge-engineering effort, serendipity of results, learns market segments	Requires some form of rating feedback, cold start for new users and new items
Content-based	No community required, comparison between items possible	Content descriptions necessary, cold start for new users, no surprises
Knowledge-based	Deterministic recommendations, assured quality, no cold-start, can resemble sales dialogue	Knowledge engineering effort to bootstrap, basically static, does not react to short-term trends

Explicit vs. Implicit Ranking

- **Explicit Feedback**
  - **Feedback that users directly report on their interest in products**
    - e.g. star ratings for movies
    - e.g. thumbs-up/down for TV shows

- **Implicit Feedback**
  - **Feedback that indirectly reflects opinion through observing user behavior**
    - e.g. purchase history, browsing history, or search patterns

Item-Based Collaborative Filtering (IBCF)

Recommender System Algorithm with Explicit Rankings

Item Explicit Ranking

Item-Item Similarity (cosine, Pearson)
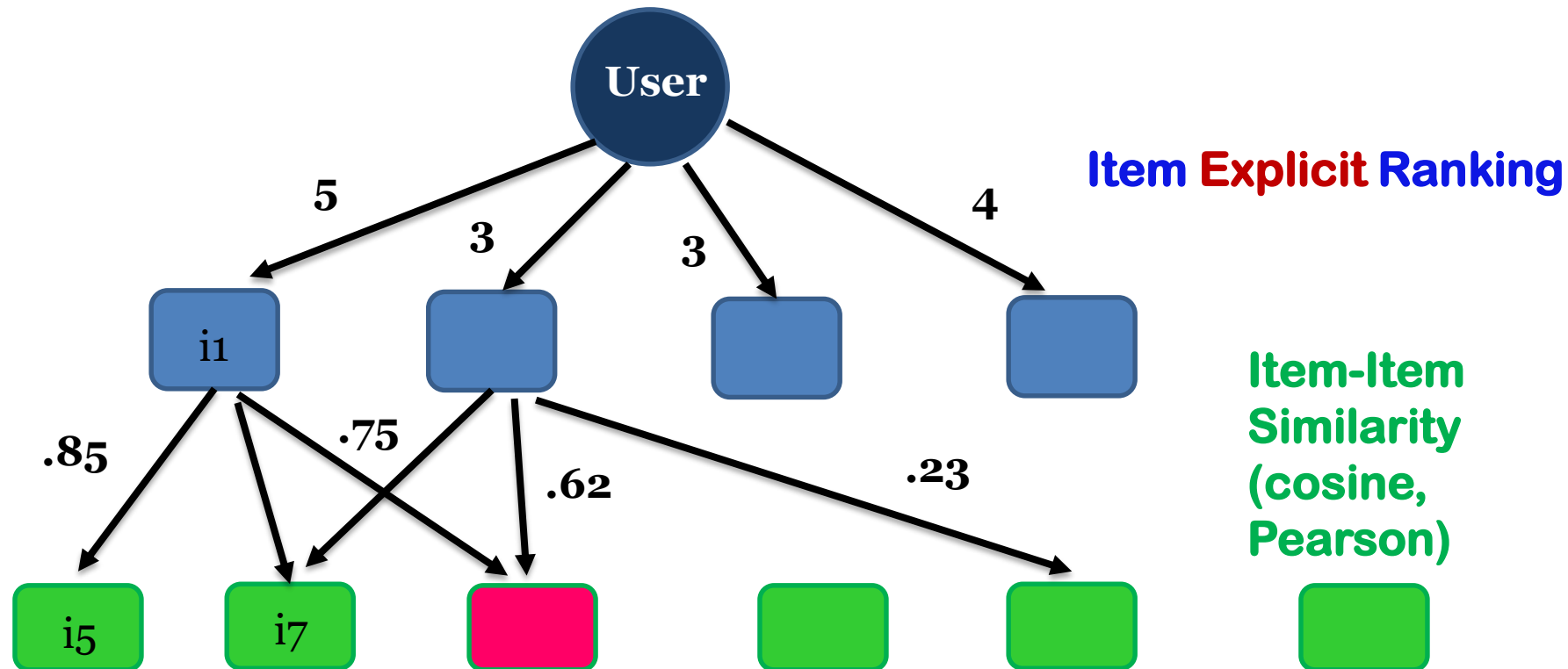
Item_Rank = Average Weighted Sum

Various Forms of Rankings

- The **goal** of a ranking system is to find the best possible ordering of a set of items for a user, within a specific context, in real-time.
- **To optimize consumption:**
  - **Rank based on *item popularity*: on average, a user is most likely to like what most others like**
  - **But, popularity is the opposite of personalization**
- **To optimize personalization**
  - **Goal: to find a personalized ranking function that is better than item popularity to better satisfy users with varying tastes.**
  - **Goal: to recommend items that each user is most likely to enjoy.**

Personalization Approach

- First, ask users to rate a few titles they have read in the past in order to build a rating prediction component.

- Then, use the user's predicted rating of each item as an adjunct to item popularity.

- Using predicted ratings on their own as a ranking function can lead to items that are too niche or unfamiliar, and can exclude items that the user would want to watch even though they may not rate them highly.

Combining Popularity w/ Personalization

- To compensate for this, rather than using either popularity or predicted rating on their own, produce rankings that balance both of these aspects: build a ranking prediction model using these two features.

What is the Goal behind Recommendation

- **What User response is a good response to a recommendation?**
  - **What are you trying to optimize when you build a RS?**
    - Not only to click but get engaged for a long time
    - The amount of time that the User engages with the service: this correlates with the business objective of retaining users over time
    - Hence, optimize over a long period of time
- **Rating prediction, or preference for each item:**
  - How much does it rely on explicit user feedback & user feedback quality? Recent trends: less explicit feedback and of the worse quality
  - Instant streaming: if you do not like smth, you will not stop to give negative feedback, you will just quickly switch to smth else
  - Assume that your actions and what you do are informing the system
  - An account that represents the whole household (not just a single user): mix of feedbacks from the household
  - Even worse: adult intentionally rate kids' content poorly so that it goes away; plus kids tend to be bi-modal in their ratings: too high or too low
  - But ton of implicit feedback

Goal of RS

- **Optimizing for next rating prediction: what is the next things you will be rating highly versus:**
  - **Rank highly documentaries but only watch them 10% of the time but 90% of the time you will watch Comedy even though your rating is not very high:**
    - Rating is low but it is entertaining and you spend lots of time on this entertaining

Ranking Problem in the Abstract Sense

- **Keep growth popularity as your baseline; good starting point when you are starting a ranking problem**

- **What can you add to this popularity to make it more personalized?**
  - What do you have at hand to help personalize the ranking system?
  - Take it as an input feature to a good ranking algorithm
  - Two features: popularity ranking and personalized ranking
  - Learning to rank: is how to learn which weights to give to each feature (what is relative merit of each of these two features: popularity vs personalized rating)

- **How to learn these weights?**
  - Treat it a classification problem: point-wise approach to learning to rank and use logistic regression (where by using a logit problem you convert a linear regression problem into a classification problem)
    - Cons: you are optimizing a log-likelihood, which is not a ranking metric
    - What you really want to do is to optimize some ranking metric that really represents your problem

Acknowledgements

- Anatoli Melechko, NCSU
- Stanford Course: http://stanford.edu/~rezab/dao/notes/lec14.pdf
- http://www.slideshare.net/srowen/big-practical-recommendations-with-alternating-least-squares
- http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/