More algorithms in Neo4j

Examples and data from OReilly's Graph Algorithms Book

(Posted in ICON)

AND https://neo4j.com/docs/graph-data-science/

Available Algorithms

Community Detection

- **Label Propagation**
- **Louvain**
- **Weakly Connected Components**
- Triangle Count
- Clustering Coefficients
- Strongly Connected Components
- Balanced Triad (identification)

Centrality / Importance

- **PageRank**
- **Personalized PageRank**
- Degree Centrality
- Closeness Centrality
- Betweenness Centrality
- ArticleRank
- Eigenvector Centrality

Similarity

- **Node Similarity**
- Euclidean Distance
- Cosine Similarity
- Overlap Similarity
- Pearson Similarity

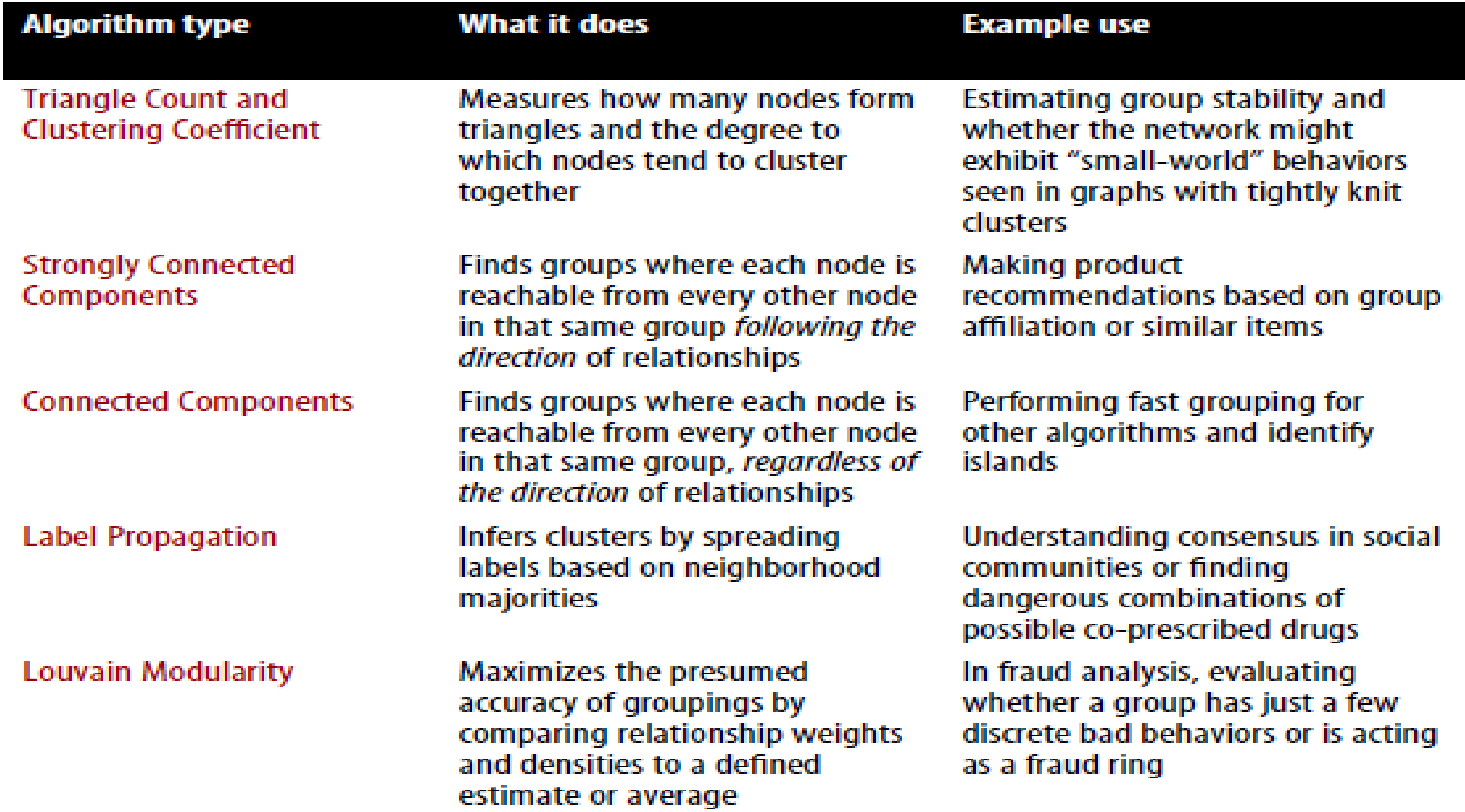
Link Prediction

- Adamic Adar
- Common Neighbors
- Preferential Attachment
- Resource Allocations
- Same Community
- Total Neighbors

Pathfinding & Search

- Parallel Breadth First Search
- Parallel Depth First Search
- Shortest Path
- Minimum Spanning Tree
- A* Shortest Path
- Yen's K Shortest Path
- K-Spanning Tree (MST)
- Random Walk

Community detection algorithms

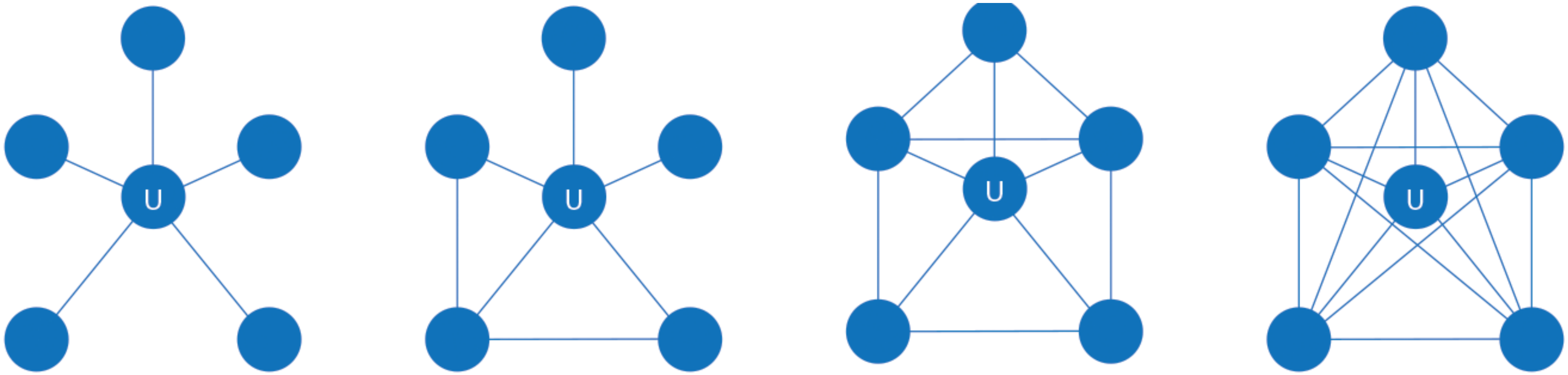


Algorithm type	What it does	Example use
Triangle Count and Clustering Coefficient	Measures how many nodes form triangles and the degree to which nodes tend to cluster together	Estimating group stability and whether the network might exhibit "small-world" behaviors seen in graphs with tightly knit clusters
Strongly Connected Components	Finds groups where each node is reachable from every other node in that same group *following the direction* of relationships	Making product recommendations based on group affiliation or similar items
Connected Components	Finds groups where each node is reachable from every other node in that same group, *regardless of the direction* of relationships	Performing fast grouping for other algorithms and identify islands
Label Propagation	Infers clusters by spreading labels based on neighborhood majorities	Understanding consensus in social communities or finding dangerous combinations of possible co-prescribed drugs
Louvain Modularity	Maximizes the presumed accuracy of groupings by comparing relationship weights and densities to a defined estimate or average	In fraud analysis, evaluating whether a group has just a few discrete bad behaviors or is acting as a fraud ring

The software graph

- The following query imports the nodes:
- **WITH** "https://github.com/neo4j-graph-analytics/book/raw/master/data/sw-nodes.csv" **AS** uri
- LOAD CSV **WITH** HEADERS FROM uri **AS** row
- MERGE (:Library {id: row.id});
- And this imports the relationships:
- **WITH** "https://github.com/neo4j-graph-analytics/book/raw/master/data/sw-relationships.csv" **AS** uri
- LOAD CSV **WITH** HEADERS FROM uri **AS** row
- **MATCH** (source:Library {id: row.src})
- **MATCH** (destination:Library {id: row.dst})
- MERGE (source)-[:DEPENDS_ON]->(destination);

Local Clustering Coefficient



Triangles = 0
Clustering Coefficient = 0
$CC(u) = \frac{0(2)}{5(5-1)}$

Triangles = 2
Clustering Coefficient = 0.2
$CC(u) = \frac{2(2)}{5(5-1)}$

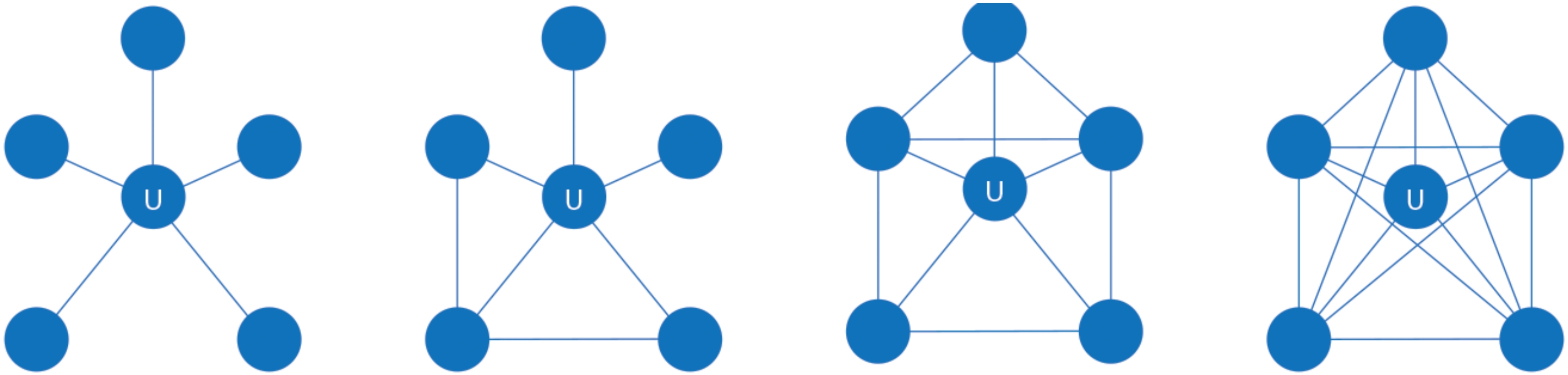
Triangles = 6
Clustering Coefficient = 0.6
$CC(u) = \frac{6(2)}{5(5-1)}$

Triangles = 10
Clustering Coefficient = 1
$CC(u) = \frac{10(2)}{5(5-1)}$

The local clustering coefficient C_n of a node *n* describes the likelihood that the neighbors of *n* are also connected. To compute C_n we use the number of triangles a node is a part of T_n, and the degree of the node d_n. The formula to compute the local clustering coefficient is as follows:

$$C_n = \frac{2T_n}{d_n(d_n - 1)}$$

Local Clustering Coefficient



Triangles = 0
Clustering Coefficient = 0

$$CC(u) = \frac{0(2)}{5(5-1)}$$

Triangles = 2
Clustering Coefficient = 0.2

$$CC(u) = \frac{2(2)}{5(5-1)}$$

Triangles = 6
Clustering Coefficient = 0.6

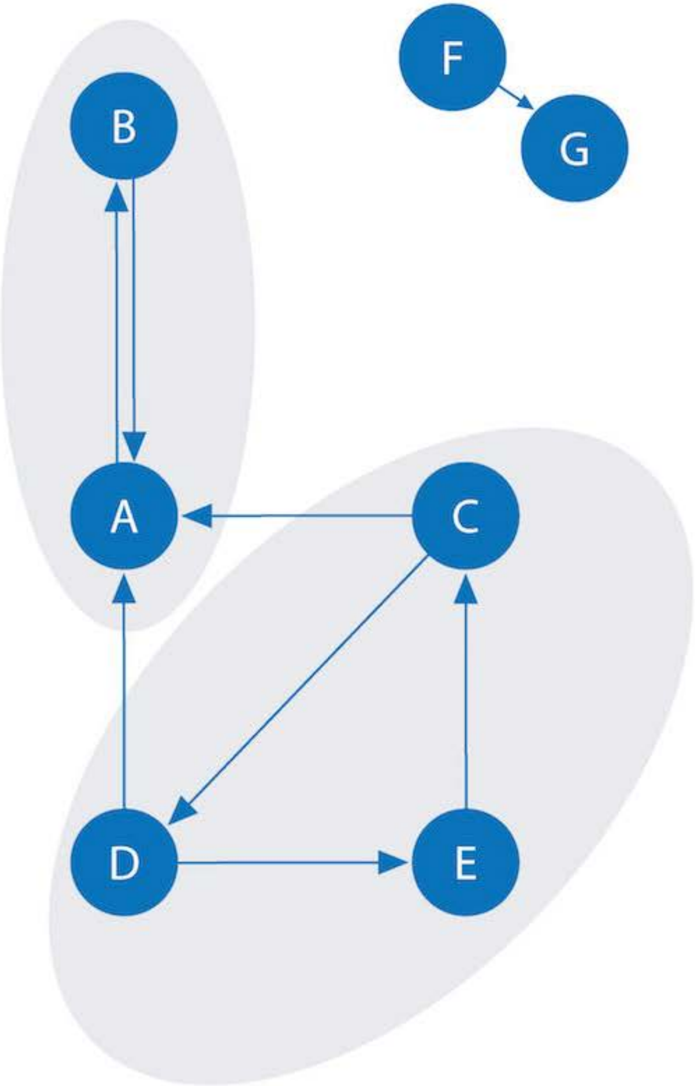
$$CC(u) = \frac{6(2)}{5(5-1)}$$

Triangles = 10
Clustering Coefficient = 1

$$CC(u) = \frac{10(2)}{5(5-1)}$$

```
CALL gds.localClusteringCoefficient.stream({
nodeProjection: "Library",
relationshipProjection: {
DEPENDS_ON: {type: "DEPENDS_ON", orientation: "UNDIRECTED“ } } })
YIELD nodeId, localClusteringCoefficient
WHERE localClusteringCoefficient > 0
RETURN gds.util.asNode(nodeId).id AS library, localClusteringCoefficient
ORDER BY  localClusteringCoefficient DESC;
```

Strongly connected components



Strongly Connected Components

Sets where all nodes can reach all other nodes in both directions, but not necessarily directly.

2 sets of strongly connected components are shown shaded : {A,B,} and {C,D,E}

Note that in {C,D,E} each node can reach the others, but in some cases they must go through another node first.

Weakly Connected components

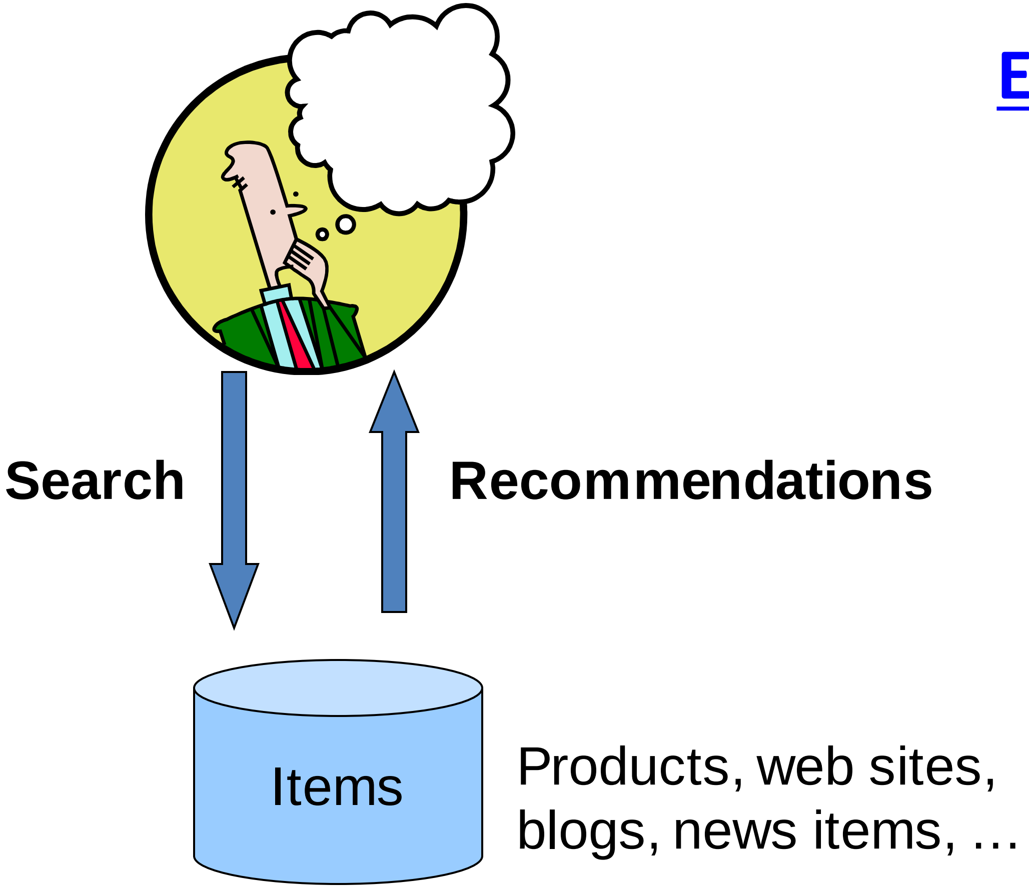
- There exist a path connecting the nodes

```
CALL gds.wcc.stream({
nodeProjection: "Library",
relationshipProjection: "DEPENDS_ON"
}) YIELD nodeId, componentId
RETURN componentId, collect(gds.util.asNode(nodeId).id) AS libraries
ORDER BY size(libraries) DESC;
```

Louvain Example

- https://neo4j.com/docs/graph-data-science/current/algorithms/louvain/

Recommendations



Search

Recommendations

Items

Products, web sites, blogs, news items, …

Examples:

amazon.com

PANDORA

StumbleUpon

del.icio.us

NETFLIX

movielens
helping you find the *right* movies

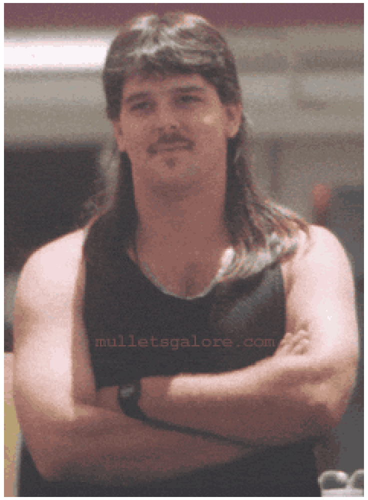
last.fm
the social music revolution

Google News

YouTube

XBOX LIVE

Example: Recommender Systems



- **Customer X**
  - Buys Metallica CD
  - Buys Megadeth CD
- **Customer Y**
  - Does search on Metallica
  - Recommender system suggests Megadeth from data collected about customer **X**

How **Into Thin Air** made **Touching the Void** a bestseller: http://www.wired.com/wired/archive/12.10/tail.html

Formal Model

- ***X*** = set of **Customers**
- ***S*** = set of **Items**

- **Utility function** ***u*: *X* × *S* → *R***
  - ***R*** = set of ratings
  - ***R*** is a totally ordered set
  - e.g., **0-5** stars, real number in **[0,1]**

Utility Matrix

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

Three approaches

- **Three approaches to recommender systems:**
  - **1)** Content-based
  - **2)** Collaborative
  - **3)** Latent factor based

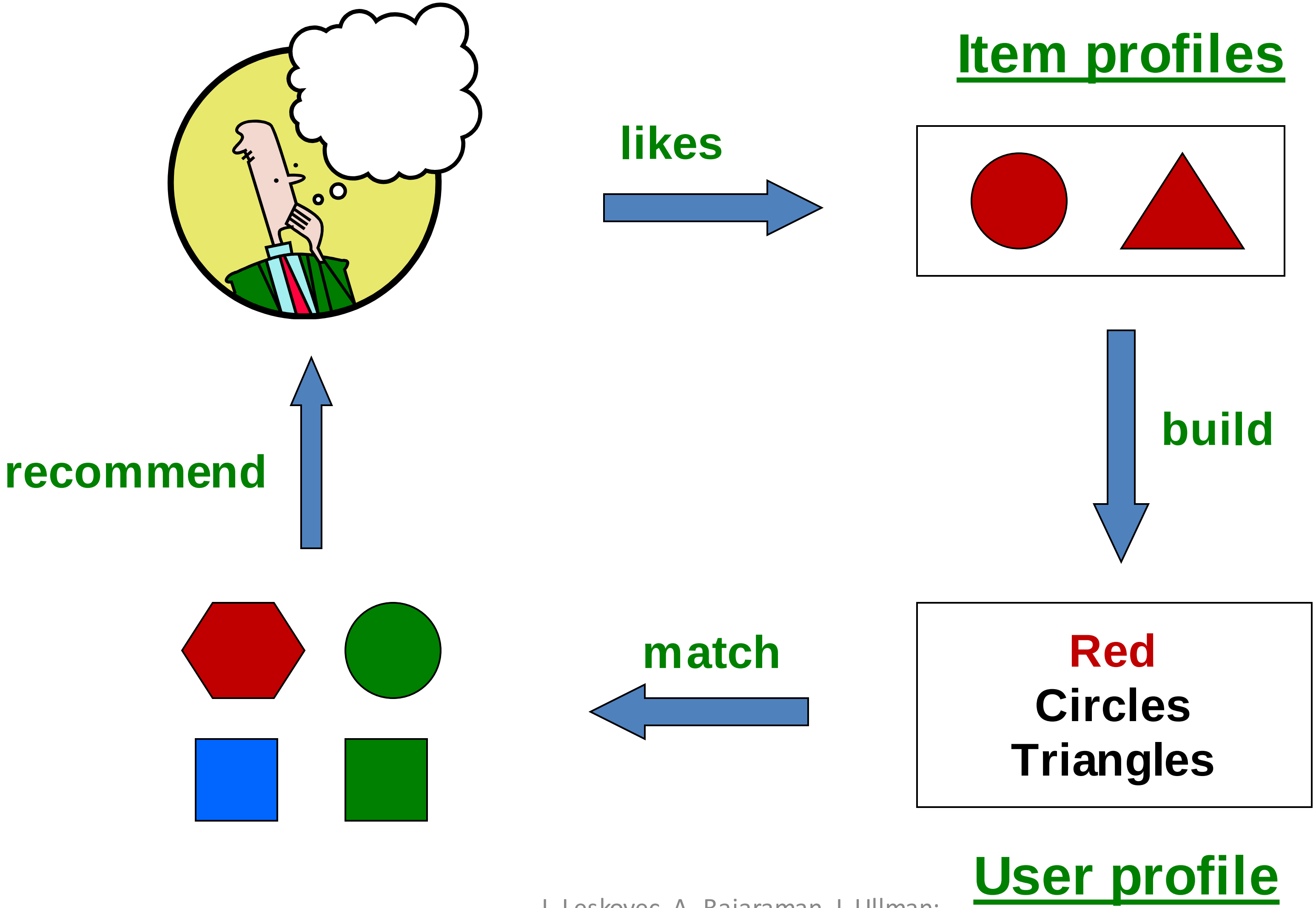
Content-based Recommendations

- **Main idea:** Recommend items to customer ***x*** similar to previous items rated highly by ***x***

Example:

- **Movie recommendations**
  - – Recommend movies with same actor(s), director, genre, …
- **Websites, blogs, news**
  - – Recommend other sites with "similar" content

Plan of Action



Item profiles

likes

build

Red
Circles
Triangles

User profile

match

recommend

Item Profiles

- For each item, create an **item profile**
- **Profile is a set (vector) of features**
  - **Movies:** author, title, actor, director,…
  - **Text:** Set of "important" words in document
- **How to pick important features?**
  - Usual heuristic from text mining is **TF-IDF** (Term frequency * Inverse Doc Frequency)

  Doc profile = set of words with highest **TF-IDF** scores, together with their scores

User Profiles and Prediction

- **User profile possibilities:**
  - Weighted average of rated item profiles
  - **Variation:** weight by difference from average rating for item
  - …
- **Prediction heuristic:**
  - Given user profile ***x*** and item profile ***i***, estimate

$$u(\boldsymbol{x}, \boldsymbol{i}) = \cos(\boldsymbol{x}, \boldsymbol{i}) = \frac{\boldsymbol{x} \cdot \boldsymbol{i}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{i}||}$$

Pros: Content-based Approach

- **+: No need for data on other users**
  - No cold-start or sparsity problems
- **+: Able to recommend to users with unique tastes**
- **+: Able to recommend new & unpopular items**
  - No first-rater problem
- **+: Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Cons: Content-based Approach

- **–: Finding the appropriate features is hard**
  - E.g., images, movies, music
- **–: Recommendations for new users**
  - **How to build a user profile?**
- **–: Overspecialization**
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - **Unable to exploit quality judgments of other users**

Collaborative Filtering

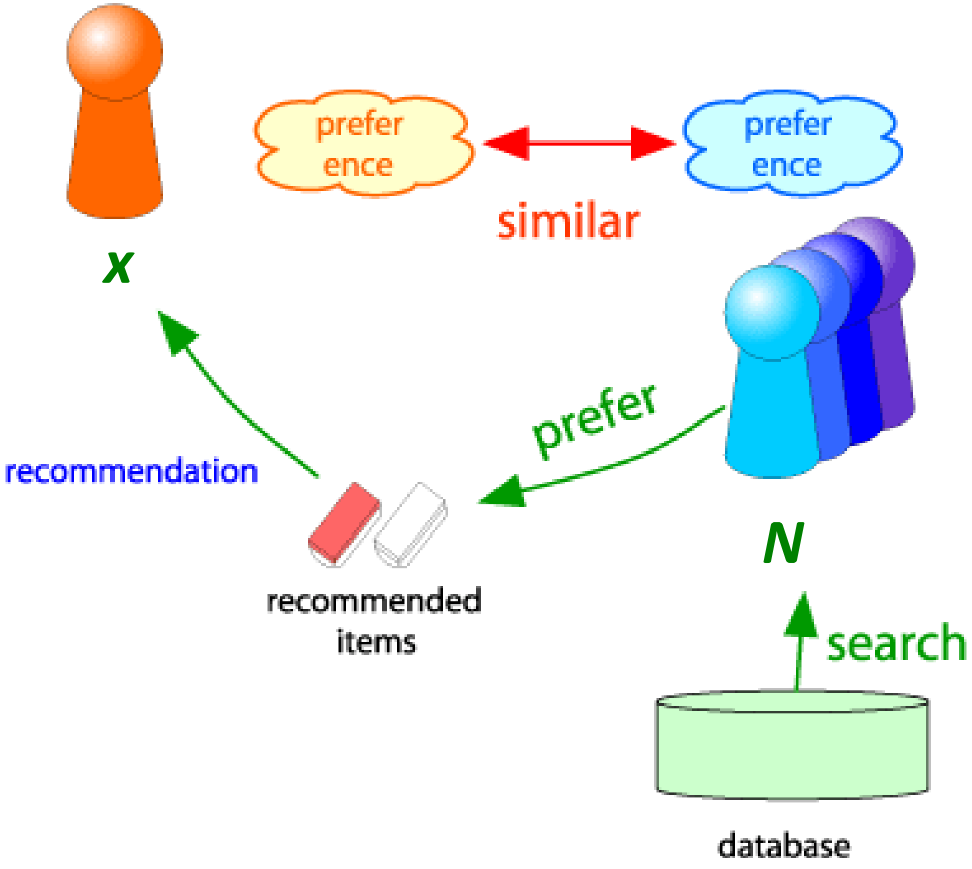
Harnessing quality judgments of other users

Collaborative Filtering

- Consider user ***x***

- Find set ***N*** of other users whose ratings are "**similar**" to ***x***'s ratings

- Estimate ***x***'s ratings based on ratings of users in ***N***



r_x = [*, _, _, *, ***]
r_y = [*, _, **, **, _]

Finding "Similar" Users

- Let r_x be the vector of user x's ratings
- **Jaccard similarity measure**
  - **Problem:** Ignores the value of the rating

r_x, r_y as sets:
r_x = {1, 4, 5}
r_y = {1, 3, 4}

$\boldsymbol{r_x}$ = [*, _, _, *, ***]
$\boldsymbol{r_y}$ = [*, _, **, **, _]

Finding “Similar” Users

- Let $\boldsymbol{r_x}$ be the vector of user ***x***’s ratings
- **Jaccard similarity measure**
  - **Problem:** Ignores the value of the rating
- **Cosine similarity measure**
  - sim(***x***, ***y***) = cos($\boldsymbol{r_x}$, $\boldsymbol{r_y}$) = $\frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$
  - **Problem:** Treats missing ratings as “negative”
    - Solution: only consider items rated by both x and y
    - Solution: subtract the (row) mean (cosine == correlation)

***r_x, r_y* as sets:**
$\boldsymbol{r_x}$ = {1, 4, 5}
$\boldsymbol{r_y}$ = {1, 3, 4}

***r_x, r_y* as points:**
$\boldsymbol{r_x}$ = {1, 0, 0, 1, 3}
$\boldsymbol{r_y}$ = {1, 0, 2, 2, 0}

r_x = [*, _, _, *, ***]
r_y = [*, _, **, **, _]

Finding “Similar” Users

- Let r_x be the vector of user ***x***’s ratings
- **Jaccard similarity measure**
  - **Problem:** Ignores the value of the rating
- **Cosine similarity measure**
  - sim(***x***, ***y***) = cos(r_x, r_y) = $\frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$
  - **Problem:** Treats missing ratings as “negative”
- **Pearson correlation coefficient**
  - S_{xy} = items rated by both users ***x*** and ***y***

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

r_x, r_y as sets:
r_x = {1, 4, 5}
r_y = {1, 3, 4}

r_x, r_y as points:
r_x = {1, 0, 0, 1, 3}
r_y = {1, 0, 2, 2, 0}

$\overline{\mathbf{r}_\mathbf{x}}$, $\overline{\mathbf{r}_\mathbf{y}}$ … avg.
rating of **x**, **y**

Rating Predictions

From similarity metric to recommendations:

- Let $\boldsymbol{r_x}$ be the vector of user ***x***'s ratings
- Let ***N*** be the set of ***k*** users most similar to ***x*** who have rated item ***i***
- **Prediction for item *s* of *user x*:**
  - $r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$
  - $r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$
  - Other options?

Shorthand:
$\boldsymbol{s_{xy} = sim(x, y)}$

- **Many other tricks possible…**

Item-Item Collaborative Filtering

- **So far: User-user collaborative filtering**
- **Another view: Item-item**
  - For item ***i***, find other similar items
  - Estimate rating for item ***i*** based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model

$$\boxed{r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}}$$

s_{ij}… similarity of items ***i*** and ***j***
r_{xj}…rating of user ***u*** on item ***j***
N(i;x)… set items rated by ***x*** similar to ***i***

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

□ - unknown rating 🟨 - rating between 1 to 5

Item-Item CF (|N|=2)

users

movies

	1	2	3	4	**5**	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

🟥 - estimate rating of movie **1** by user **5**

Item-Item CF (|N|=2)

users

movies

	1	2	3	4	**5**	6	7	8	9	10	11	12	**sim(1,m)**
1	1		3		?	5			5		4		**1.00**
2			5	4			4			2	1	3	**-0.18**
3	2	4		1	2		3		4	3	5		**0.41**
4		2	4		5			4			2		**-0.10**
5			4	3	4	2					2	5	**-0.31**
6	1		3		3			2			4		**0.59**

Neighbor selection:
Identify movies similar to
movie **1**, **rated by user 5**

Here we use Pearson correlation as similarity:

1) Subtract mean rating m_i from each movie i

$m_1 = (1+3+5+5+4)/5 = \mathbf{3.6}$

row 1: *[-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]*

2) Compute cosine similarities between rows

Item-Item CF (|N|=2)

users

movies

	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Compute similarity weights:

$s_{1,3}$=0.41, $s_{1,6}$=0.59

Item-Item CF (|N|=2)

users

movies

	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		**2.6**	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Predict by taking weighted average:

$r_{1.5}$ = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6

$$r_{ix} = \frac{\sum_{j\in N(i;x)} s_{ij}\cdot r_{jx}}{\sum s_{ij}}$$

Before:

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

CF: Common Practice

- Define **similarity** ***s***s_{ij} of items ***i*** and ***j***
- Select ***k*** nearest neighbors ***N(i; x)***
  - Items most similar to ***i***, that were rated by ***x***
- Estimate rating r_{xi} as the weighted average:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$\boldsymbol{b_{xi} = \mu + b_x + b_i}$$

- μ = overall mean movie rating
- b_x = rating deviation of user ***x***
  = *(avg. rating of user* ***x****)* **–** μ
- b_i = rating deviation of movie ***i***

Baseline predictor

$$r_{xi} = \underbrace{\mu}_{\text{Overall mean rating}} + \underbrace{b_x}_{\text{Bias for user } x} + \underbrace{b_i}_{\text{Bias for movie } i}$$

- **Example:**
  - Mean rating: $\boldsymbol{\mu = 3.7}$
  - You are a critical reviewer: your ratings are 1 star lower than the mean: $\boldsymbol{b_x = -1}$
  - Star Wars gets a mean rating of *0.5* higher than average movie: $\boldsymbol{b_i = + 0.5}$
  - Predicted rating for you on Star Wars:
    = 3.7 - 1 + 0.5 = 3.2

Item-Item vs. User-User

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.8	
Bob		0.5		0.3
Carol	0.9		1	0.8
David			1	0.4

- **In practice, it has been observed that item-item often works better than user-user**
- **Why?** Items are simpler, users have multiple tastes

Pros/Cons of Collaborative Filtering

- **+ Works for any kind of item**
  - No feature selection needed
- **- Cold Start:**
  - Need enough users in the system to find a match
- **- Sparsity:**
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items
- **- First rater:**
  - Cannot recommend an item that has not been previously rated
  - New items, Esoteric items
- **- Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

For today…

- Follow the cosine similarity example: https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/cosine/
- Add yourself as a node and include your cuisine preferences.
- Write a query that returns the 3 most similar persons to yourself, sorted by similarity
- (optionally) Write a query that would connect the 3 most similar persons to you with a relationship MYSIMILARITY and the similarity as the weight
- Upload the result to ICON