3. Recommender systems

Problem: What items to recommend to which user? Products, music, movies, dishes, learning material,...

Image source `https://sudonull.com/post/12374-Anatomy-of-recommendation-systems-Part-one`

Data and utility matrix

Data: User profiles, product descriptions, browsing and buying behaviour, explicit ratings.

Often possible to derive a **utility matrix $\mathbf{A}$**, where $\mathbf{A}[i, j]$ = utility of item j for user i

- $n \times d$ matrix, n=number users, d=number of items

Two types:

1. Only positive preferences ("likes", browsing, buying)
2. Positive and negative preferences ("likes" and "dislikes", ratings)

- extremely large and sparse matrices!

Example utility matrices (movie preferences)

	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS	SCARFACE	SPARTACUS
U_1	1			5		2
U_2		5			4	
U_3	5	3		1		
U_4			3			4
U_5				3	5	
U_6	5		4			

(a) Ratings-based utility

	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS	SCARFACE	SPARTACUS
U_1	1			1		1
U_2		1			1	
U_3	1	1		1		
U_4			1			1
U_5				1	1	
U_6	1		1			

(b) Positive-preference utility

Empty cell=unspecified; in data, e.g., –, na, 0 (if non-positive ratings).

Image source Aggarwal Ch 18

Main approaches: Content-based and preference-based (collaborative filtering)

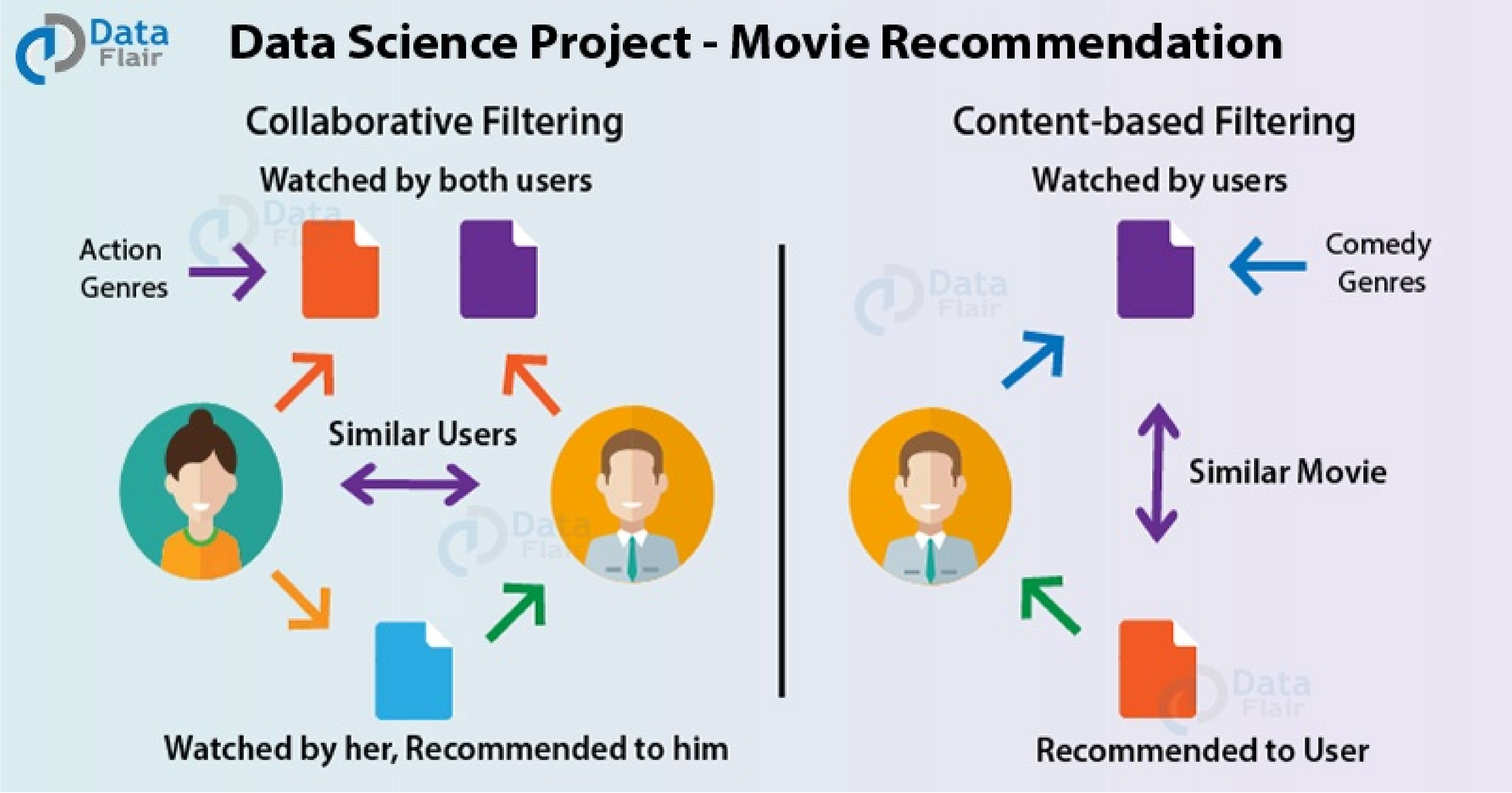


Image source
https://data-flair.training/blogs/data-science-r-movie-recommendation/

Content-based recommendations

Given

1. **item profile** = text descriptions, keywords
2. **user profile** = documents describing user's interests (e.g., descriptions of previously bought/liked items, explicitely specified or derived interests)

Search items whose profiles match (are similar) to the user's profile

a) **If no utility matrix**

- search K most similar items to the user profile
- e.g., tf-idf presentation + cos similarity

Content-based recommendations

b) **If utility matrix exists**, utilize the user's previous preferences!

= prediction task where vector $\mathbf{A}[i]$ = target values for user i

- If positive preference matrix, learn a classifier
- If numerical ratings, learn a regression model

– training sets extremely small

– over-specialization: recommendations tend to favour items described by the same keywords
    - e.g., recommend movies with the same actors as before

Collaborative filtering

Assumption: The user probably likes what other similar users have liked.

Approaches for recommendation

i) Neighbourhood-based

ii) Graph-based

iii) Clustering-based

iv) Latent factor -based

Neighbourhood-based methods: 1. user-based

Utilize **user–user similarity**

e.g., **Pearson correlation coefficient** r for similarity between two users' rating vectors $\mathbf{x} = (x_1, \ldots, x_d)$ and $\mathbf{y} = (y_1, \ldots, y_d)$:

$$r(\mathbf{x}, \mathbf{y}) = \frac{\sum_{j \in J}(x_j - \mu_x)(y_j - \mu_y)}{\sqrt{\sum_{j \in J}(x_j - \mu_x)^2 \sum_{j \in J}(y_j - \mu_y)^2}}$$

$J = \{j \mid x_j \neq na, y_j \neq na\}$ (items rated by both)
μ_x is average rating, two alternatives:

i) $\mu_x = \frac{1}{|J|} \sum_{j \in J} x_j$ (only common items) or

ii) $\mu_x = \frac{1}{|J_x|} \sum_{j \in J_x} x_j$, where $J_x = \{j \mid x_j \neq na\}$ (all rated items; more common approach)

Predict missing ratings in rating vector $\mathbf{x}$

1. search K nearest neighbours $NN_{\mathbf{x}}$ using similarity r
2. remove neighbours from $NN_{\mathbf{x}}$ if $r \leq \theta$ (negative or weak correlations)
3. normalize ratings: $y'_j = y_j - \mu_{\mathbf{y}}$ (since in different scales)
4. calculate predicted rating for all items j with missing entries in $\mathbf{x}$:

$$\tilde{x}_j = \frac{\sum_{\mathbf{y} \in NN_{\mathbf{x}}} w_{\mathbf{y}} \cdot y'_j}{\sum_{\mathbf{y} \in NN_{\mathbf{x}}} w_{\mathbf{y}}} + \mu_x$$

- $w_{\mathbf{y}} = 1$ or weigh by similarity $w_{\mathbf{y}} = r(\mathbf{x}, \mathbf{y})$
- i.e., weighted average rating by similar users + return to $\mathbf{x}$'s original scale

Example: Predict missing ratings ($K = 2$, $r \geq 0.5$)

	m_1	m_2	m_3	m_4	m_5	m_6
u_1	–	1	2	2	3	–
u_2	3	1	1	2	4	3
u_3	4	2	3	3	–	5
u_4	2	5	4	–	1	2

User means:
$\mu_1 = 2.000$
$\mu_2 = 2.333$
$\mu_3 = 3.400$
$\mu_4 = 2.800$

	u_1	u_2	u_3	u_4
u_1	1.000	0.836	0.927	-0.917
u_2	0.836	1.000	0.822	-0.974
u_3	0.927	0.822	1.000	-0.862
u_4	-0.917	-0.974	-0.862	1.000

m_1=Gladiator, m_2=Godfather, m_3=Ben-Hur, m_4=Goodfellas, m_5=Scarface, m_6=Spartacus

Example: Predict missing ratings ($K = 2$, $r \geq 0.5$)

	u_1	u_2	u_3	u_4
u_1	1.000	0.836	0.927	-0.917
u_2	0.836	1.000	0.822	-0.974
u_3	0.927	0.822	1.000	-0.862
u_4	-0.917	-0.974	-0.862	1.000

$\mu_1 = 2.000$
$\mu_2 = 2.333$
$\mu_3 = 3.400$
$\mu_4 = 2.800$

for u_1 nearest u_3 and u_2, predicted for u_1, m_1:

$\frac{0.836 \cdot (3-2.333) + 0.927 \cdot (4-3.400)}{0.836+0.927} + 2.000 = 2.63 > \mu_1 \rightarrow$ recommend

for u_1, m_6 predicted 3.16
for u_3 nearest u_1 and u_2, for m_5 predicted 4.71
for u_4 not enough neighbours! (all $r < 0$)

Neighbourhood-based methods: 2. item-based

Utilize **item–item similarity**

$\mathbf{v}$ = jth item's rating vector, $\mathbf{x}$ = ith user's rating vector

1. search K nearest neighbours $NN_{\mathbf{v}}$ of $\mathbf{v}$
2. select a subset $NN_{\mathbf{v},\mathbf{x}} \subseteq NN_{\mathbf{v}}$ of those items's ratings that user i has rated: $NN_{\mathbf{v},\mathbf{x}} = \{\mathbf{u}_r \mid \mathbf{u}_r \in NN_{\mathbf{v}}, x_r \neq na\}$
3. Predicted rating is

$$\tilde{x}_j = \frac{\sum_{\mathbf{u}_r \in NN_{\mathbf{v},\mathbf{x}}} w_{\mathbf{v},\mathbf{u}_r} \cdot x_r}{\sum_{\mathbf{u}_r \in NN_{\mathbf{v},\mathbf{x}}} w_{\mathbf{v},\mathbf{u}_r}}$$

- i.e., weighted average rating on similar items by user i

Neighbourhood-based methods: 2. item-based

What similarity measure to use? Should we normalize ratings?

- Pearson correlation (+ mean centering can be used also here)
- **adjusted cosine similarity** = cosine similarity after mean centering each user's ratings
- **Problem**: cosine similarity with 0 vectors (movies with average ratings from everybody) is not defined $\leftrightarrow$ cos works better when all values positive

See Aggarwal 18.5.2.2

Graph-based methods

Idea: Create a bipartite **user-item graph** and utilize random walk approaches.

- graph $\mathbf{G} = (\mathbf{U} \cup \mathbf{V}, \mathbf{E})$
- $\mathbf{U}$ = nodes for users
- $\mathbf{V}$ = nodes for items
- $\mathbf{E}$ = edges such that $(u_i, v_j) \in \mathbf{E}$, $u_i \in \mathbf{U}$, $v_j \in \mathbf{V}$, if the ith user has rated the jth item
- if rating matrix (positive and negative preferences), the edges may have weights:
  - normalize rating $\mathbf{A}[i, j]$ by subtracting mean of ratings on row $\mathbf{A}_i$ $\rightarrow$ signed network

Bipartite user-item graph

	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS	SCARFACE	SPARTACUS
U_1	1			1		1
U_2		1			1	
U_3	1	1		1		
U_4			1			1
U_5				1	1	
U_6	1		1			

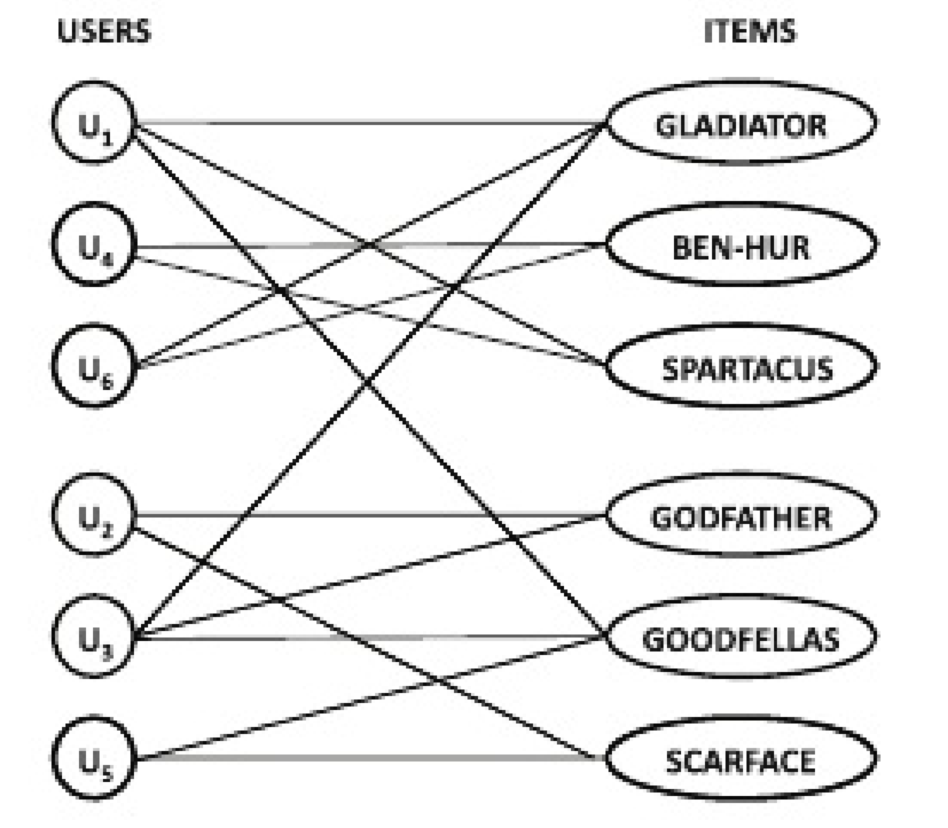


Image source Aggarwal Ch 18

Making recommendations

Let $\mathbf{G}$ be unweighted (presents only positive preferences).

Two approaches:

1. Use G only to determine nearest neighbours:

- determine K most similar users to the ith user using personalized PageRank or SimRank
- or K most similar items to the jth item
- make recommendations as before (user-based or item-based)

Making recommendations

2. Use PageRank values to decide recommendations:

i) given user i, search **item nodes** with largest PageRank values, when teleportation to user node u_i
$\rightarrow$ recommend these items to the ith user

ii) given item j, search **user nodes** with largest PageRank values, when teleportation to item node v_j
$\rightarrow$ recommend the jth item to these users

- teleportation probability α affects results
  - small α favours popular items
  - larger α makes recommendations more specific to the given user

Clustering-based methods

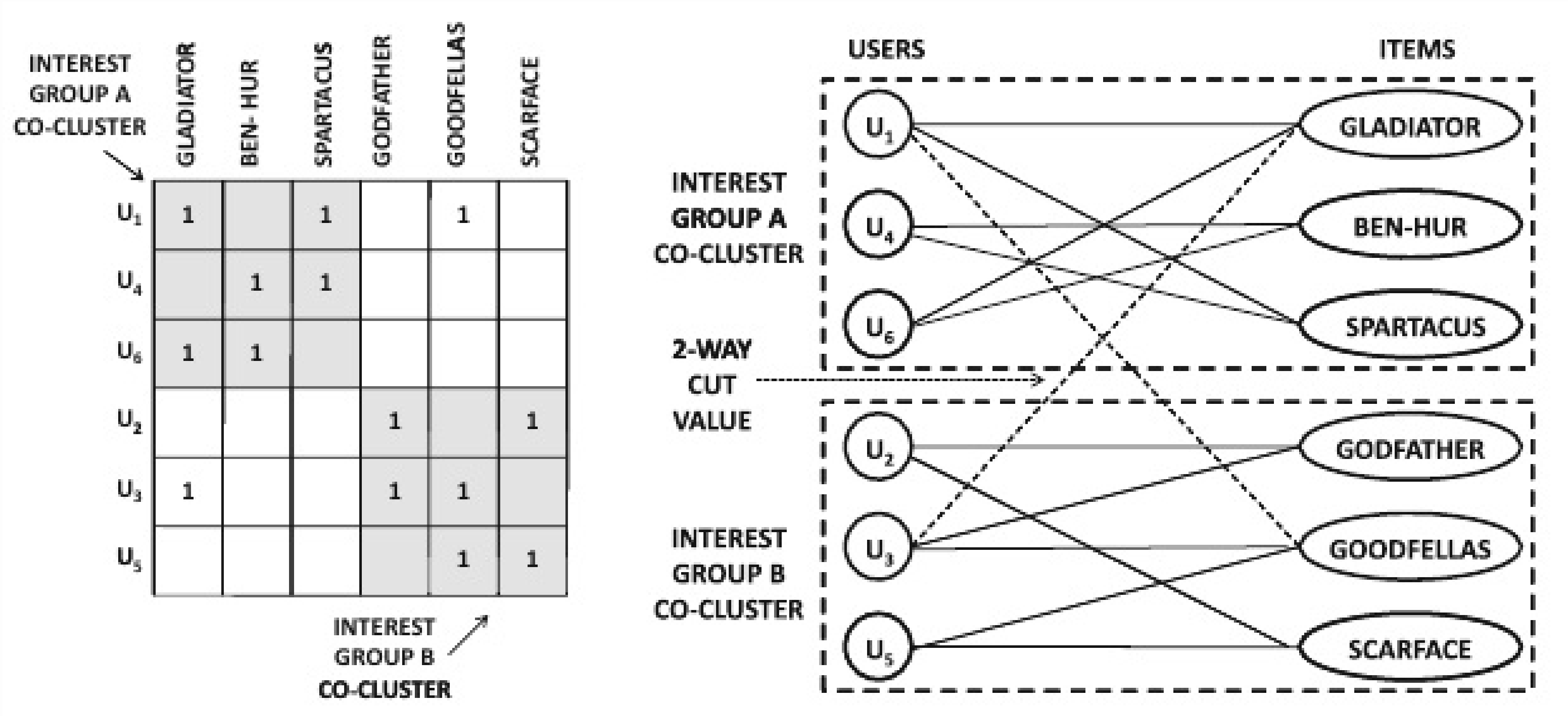
Idea: Determine peer groups (similar users or similar items) beforehand by clustering.

$\leftrightarrow$ neighbourhood-based methods determine them separately for all users

What clustering methods to use?

- problem: data sets very sparse (many missing values)
- adapt K-means:
  - calculate distances $d(\mathbf{x}, \mathbf{c}_i)$ and centroids $\mathbf{c}_i$ only over those dimensions where ratings available
- co-clustering approaches

Co-clustering of movie preference data



(a) Co-cluster (b) User-item graph

Image source Aggarwal Ch 18

Latent factor -based methods

Idea: summarize correlations by latent factors $\rightarrow$ smaller dimensional representation of utility matrix $\mathbf{A} \approx \mathbf{F}_U\mathbf{F}_I^T$

- present n users by n k-dimensional latent factors, $\mathbf{F}_{U1}, \ldots, \mathbf{F}_{Un}$
- present d items by d k-dimensional latent factors, $\mathbf{F}_{I1}, \ldots, \mathbf{F}_{In}$
- k = new reduced dimensionality of latent representation
- estimate rating $\mathbf{A}[i, j] \approx \mathbf{F}_{Ui} \cdot \mathbf{F}_{Ij}$
- use (modified) SVD or other matrix factorization to get latent factors

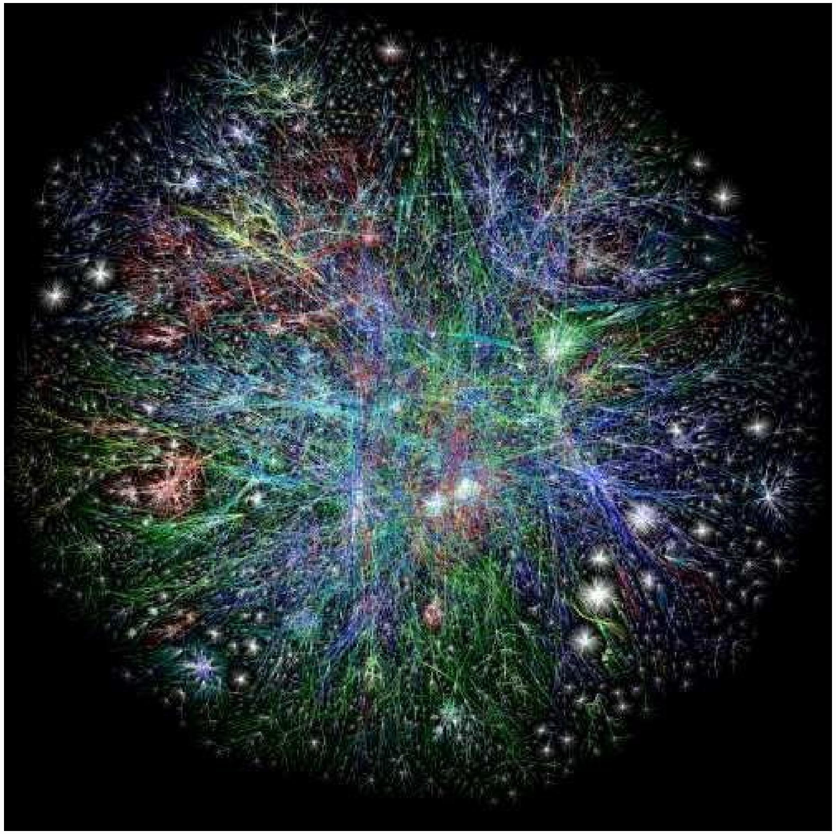
Further reading Aggarwal 18.5.5 and 6.8

Summary

- Content-based recomendations: evaluate similarity between text descriptions and utilize only the user's own ratings
- Collaborative filtering: utilize all users' ratings
  - many approaches: neighbourhood-based, graph-based, clustering-based, latent factor-based
  - often 2 steps: determine a peer group and then calculate predicted ratings
  - sometimes 1 step: choose recommended items directly by PageRank or use matrix factorization

Further reading: Desrosiers and Karypis: A comprehensive survey of neighborhood-based recommendation methods. In Recommender Systems Handbook, 2011.

Mining graph data: Web and recommender systems



1. Introduction to Graph mining
2. Web mining and search
   - Emphasis: **Ranking algorithms**
3. Recommender systems
   - Emphasis: **Collaborative filtering**

image source: Opte project, `https://www.opte.org/the-internet`

1. Introduction to Graph mining

Data may consist of

1. one large graph
   - e.g., internet, social network
2. multiple small graphs
   - e.g., chemical compounds, biological pathways, program control flows, consumer behaviour, ...

Information to mine: interesting substructures, influential nodes, similarities, communities, clusters

Data: one large graph

Internet (Wikipedia hyperlink structure)

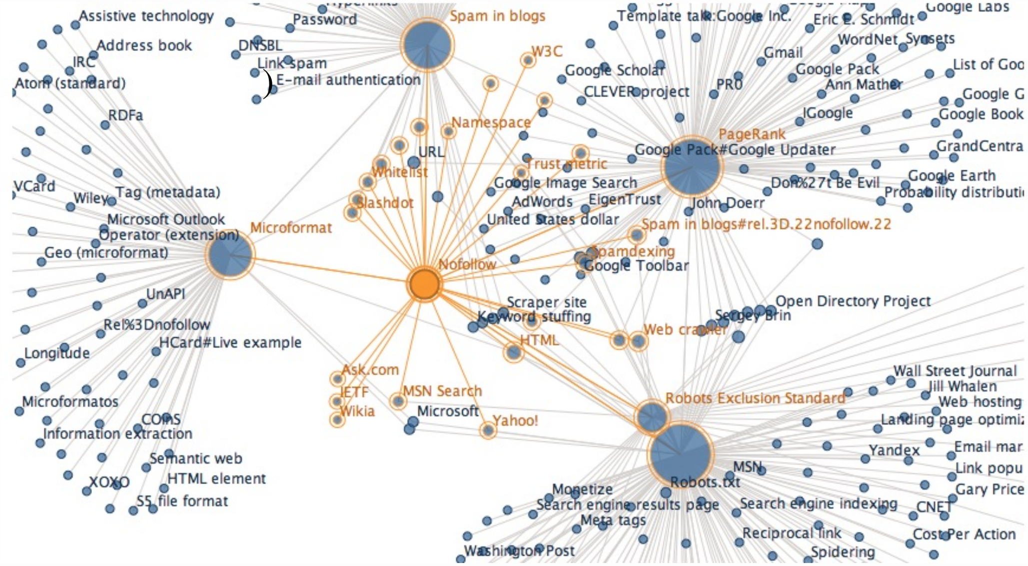


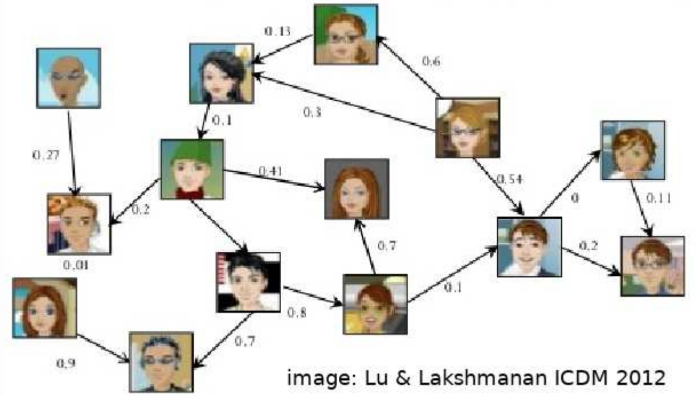
Image source:https://wiki.digitalmethods.net/Dmi/WikipediaAnalysis

Most influential nodes?

Communities or dense subgraphs?

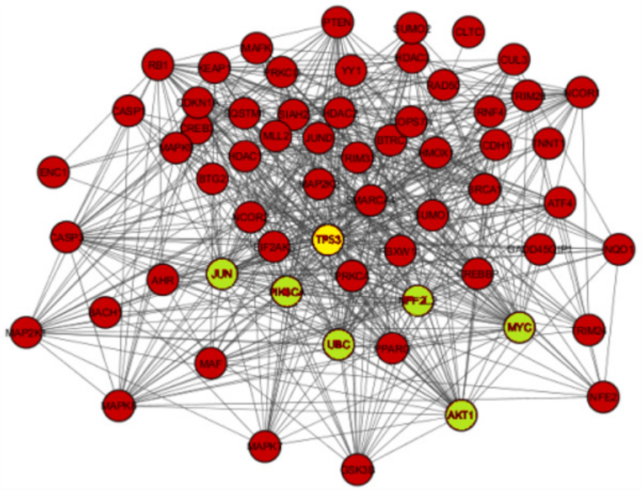
Node similarity/future links?

Social network



https://www.slideshare.net/WeiLu12/profit–maximization–over–social–networks

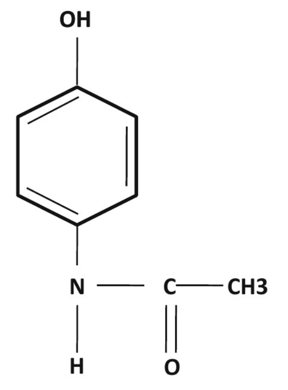
Protein–protein interaction network

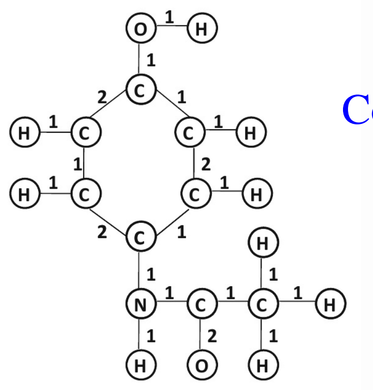


Srivastava et al. (2018)
doi 10.2174/1875036201811010240

Data: multiple (smaller) graphs

Molecular structure graphs





(a) Acetaminophen (b) Graph representation

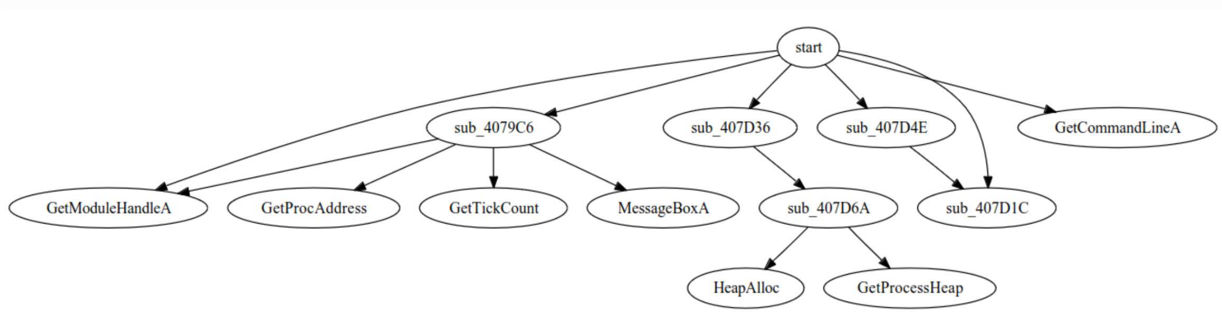
Aggarwal Fig. 17.1

Common and unique properties?

Frequent subgraphs?

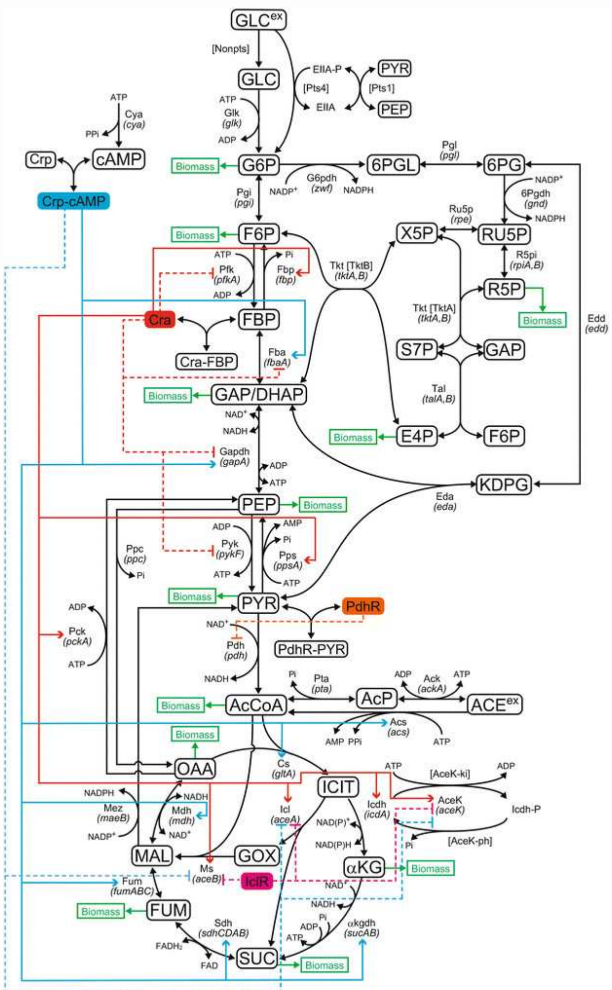
Clusters?

Software call graphs (malware)



Kinable & Kostakis (2011)
doi https://doi.org/10.1007/s11416−011−0151−y

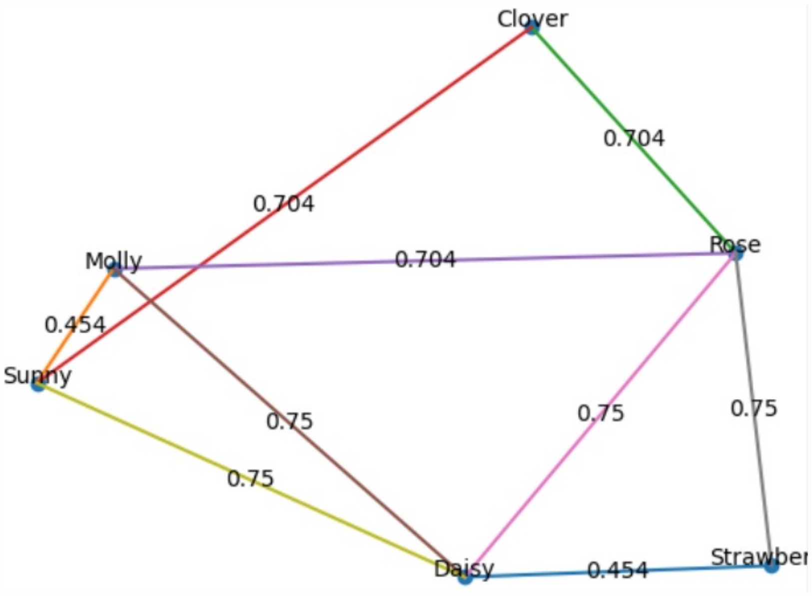
Metabolic ntwork (E. coli)



Jahan et al. (2016)
DOI:10.1186/s12934−016−0511−x

Data: graph-form presentation of other data

Similarity graph



User-item utility graph

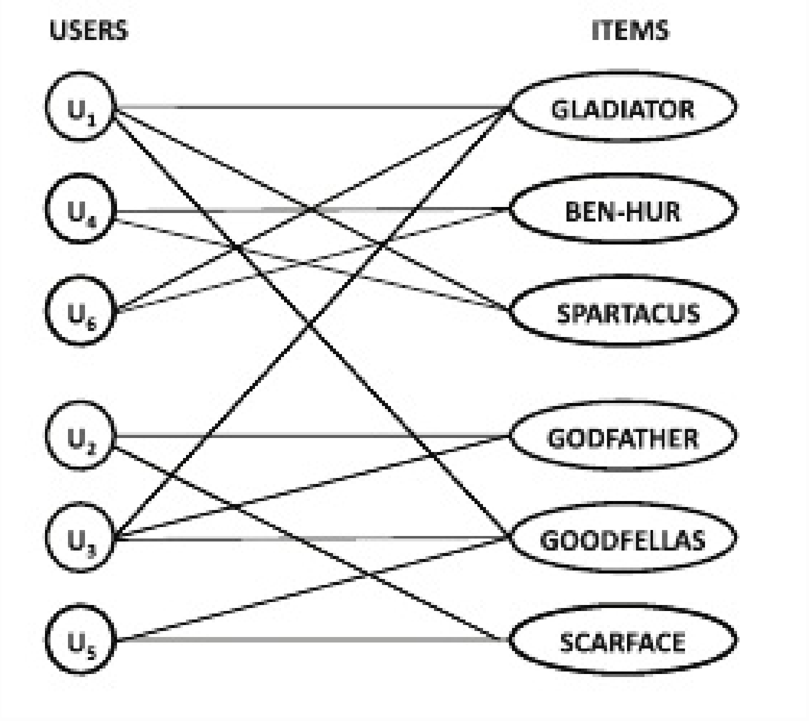


Image sources MDM2022/Laaksonen and Aggarwal Fig 18.5

2a Web mining

Data:

1. Web content = documents and links
2. Wed usage data, like buying and browsing behaviour, ratings, logs

Applications:

1. **Content-centric**: document clustering, resource discovery, web search, linkage mining
2. **Usage-centric**: recommender systems, web log analysis

2b Web search (information retrieval)

1. **Web crawling**: collect all (relevant) documents into a central location (+ keep it up to date) $\rightarrow$ Aggarwal 18.2
2. **Index construction** for the collection (offline)
3. **Query processing** (online)
   - query = set of keywords $\mathbf{q} = (w_1, \ldots, w_k)$
   - identify documents containing all/most $w_i \in \mathbf{q}$
   - **rank** documents (relevance to the query and quality)
   - return best ranked documents

Search engine: steps 2 + 3

Index construction

Preprocess documents, extract relevant tokens (words or phrases) and construct **inverted indices**

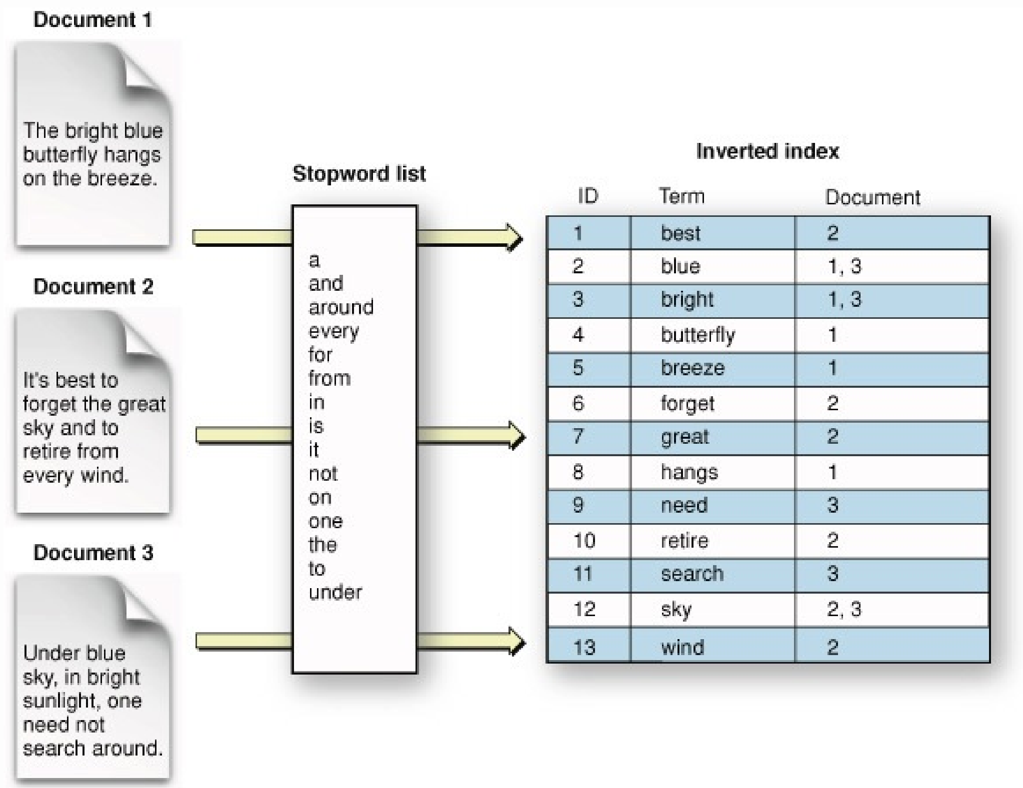


Image source https://community.hitachivantara.com/s/article/search-the-inverted-index

Ranking documents

Combine two types of scores:

1. Content-based scores based on occurrence of keywords

- frequency of word
- where the word occurs (more weight if in the title or anchor text)
- prominence of the word (font size, colour)
- relative position of keywords (more relevant if close to each other)

How web spammer can cheat the search engine to get high content-based scores?

Ranking documents

2. Reputation-based scores utilize natural voting mechanisms

a) assumptions based on page citations

- High quality pages are pointed by many pages or
- High quality pages are pointed by many high quality pages

b) user feedback

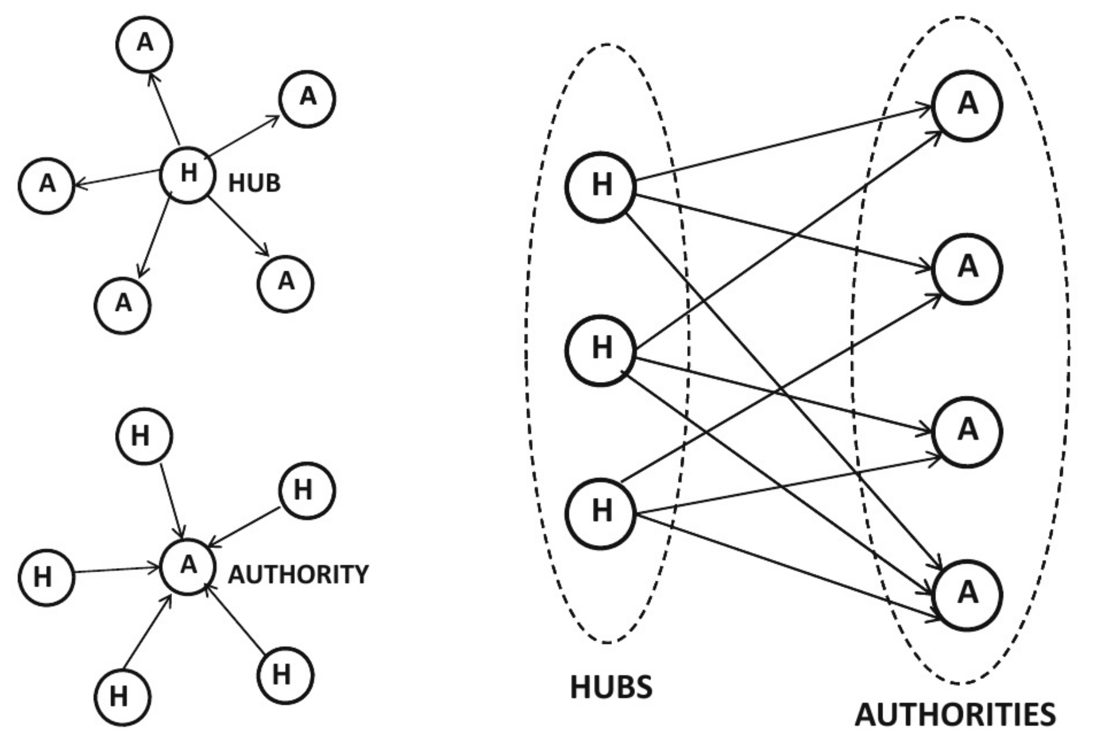
- users choose relevant pages
- return other similar pages or pages accessed by similar users (recommender systems)

Two ranking algorithms

1. HITS = Hyperlink-Induced Topic Search =
   ”Hubs and Authorities algorithm”
   - query-dependent
   - There are two types of good pages: good authority pages are reputable sources on topics and good hub pages offer links to good sources.
2. PageRank
   - query-independent
   - Highly reputabe pages are likely cited by other higly reputable pages.

Hyperlink-Induced Topic Search (HITS)

hub = page that refers to many authoritative pages
authority = authoritative page refered by many hubs



(a) Hub and authority examples (b) Network organization between hubs and authorities

image source Aggarwal Fig. 18.3

HITS basic idea

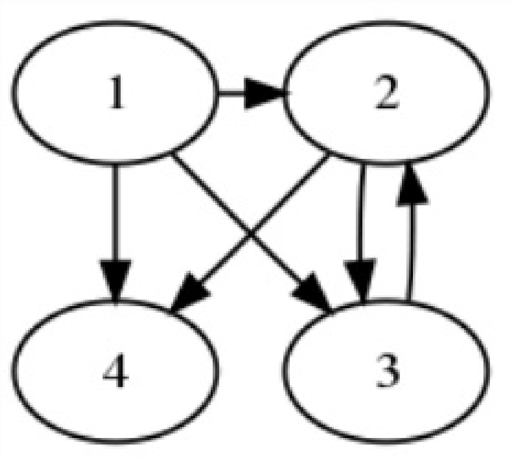
1. Construct an input graph:
   - search top-K pages most relevant to the query (e.g., top-200) $\rightarrow$ root set $\mathbf{R}$
   - add pages pointed by $v \in \mathbf{R}$ and some of pages pointing to $v \in \mathbf{R}$ (e.g., max 50/per node) $\rightarrow$ base set $\mathbf{V}$
   - construct a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ where $\mathbf{E}$ contains all links between nodes in $\mathbf{V}$
2. Assign all nodes hub and authority weights, $h(v)$ and $a(v)$
   - update weights until the system converges
   - return nodes v with highest $a(v)$

HITS algorithm on $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

$In(v) = \{u \mid (u, v) \in \mathbf{E}\}$ nodes pointing to v
$Out(v) = \{u \mid (v, u) \in \mathbf{E}\}$ nodes to which v points

1. initialize h and a such that $\sum_i h(v_i)^2 = \sum_i a(v_i)^2 = 1$ (e.g., $h(v_i) = a(v_i) = \frac{1}{\sqrt{n}}$)
2. until h and a values converge, update for all v_i:
   - $h(v_i) = \sum_{w \in Out(v_i)} a(w)$ (authorities pointed by v_i)
   - $a(v_i) = \sum_{w \in In(v_i)} h(w)$ (hubs pointing to v_i)
   - normalize weights: $h(v_i) = \frac{h(v_i)}{H}$ and $a(v_i) = \frac{a(v_i)}{A}$, where $H = \sqrt{\sum_i h(v_i)^2}$ ja $A = \sqrt{\sum_i a(v_i)^2}$
3. return v_is with largest $a(v_i)$

Example: calculation of h and a



initialize $h(i) = a(i) = \frac{1}{\sqrt{4}} = \frac{1}{2}$

round 1:

$h(1) = \frac{3}{2}$, $a(1) = 0$

$h(2) = \frac{2}{2}$, $a(2) = \frac{2}{2}$

$h(3) = \frac{1}{2}$, $a(3) = \frac{2}{2}$

$h(4) = 0$, $a(4) = \frac{2}{2}$

normalize by $H = \sqrt{\frac{7}{2}}$ and $A = \sqrt{3}$

$h(1) = \frac{3}{\sqrt{14}}$, $a(1) = 0$

$h(2) = \frac{2}{\sqrt{14}}$, $a(2) = \frac{1}{\sqrt{3}}$

$h(3) = \frac{1}{\sqrt{14}}$, $a(3) = \frac{1}{\sqrt{3}}$

$h(4) = 0$, $a(4) = \frac{1}{\sqrt{3}}$

Example: final result

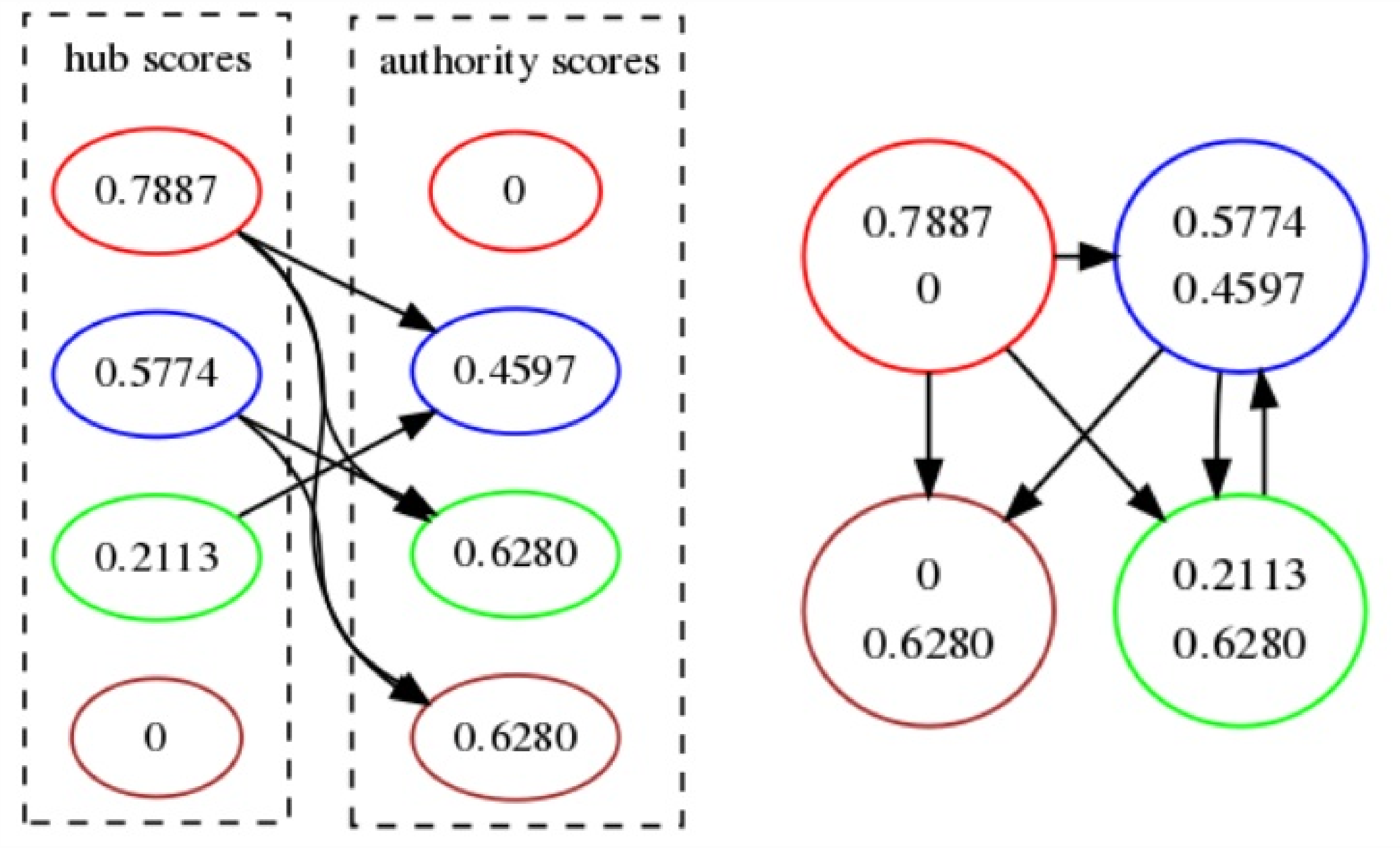


image source Pajarinen (2008): The PageRank/HITS algorithms (slides)

PageRank

Uses a **random surfer model = random walk model**

- visit random pages by selecting random links
- long term visiting frequency of a page depends on the number of in-linking pages and their visiting frequency
- $\Rightarrow$ frequency is high, if linked from other frequently visited pages
- reputation $\approx$ long term frequency of visits by a random surfer = **steady-state probability** π

$\Rightarrow$ calculate $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)^T$, where π_i = PageRank of the ith node v_i (n=number of pages)

Problem: the surfer can get trapped!

Originally, all out-going links from a node have equal probability.

Dead-end nodes don't have out-going links

$\Rightarrow$ add links from a dead-end node to all n pages with transition probability $\frac{1}{n}$

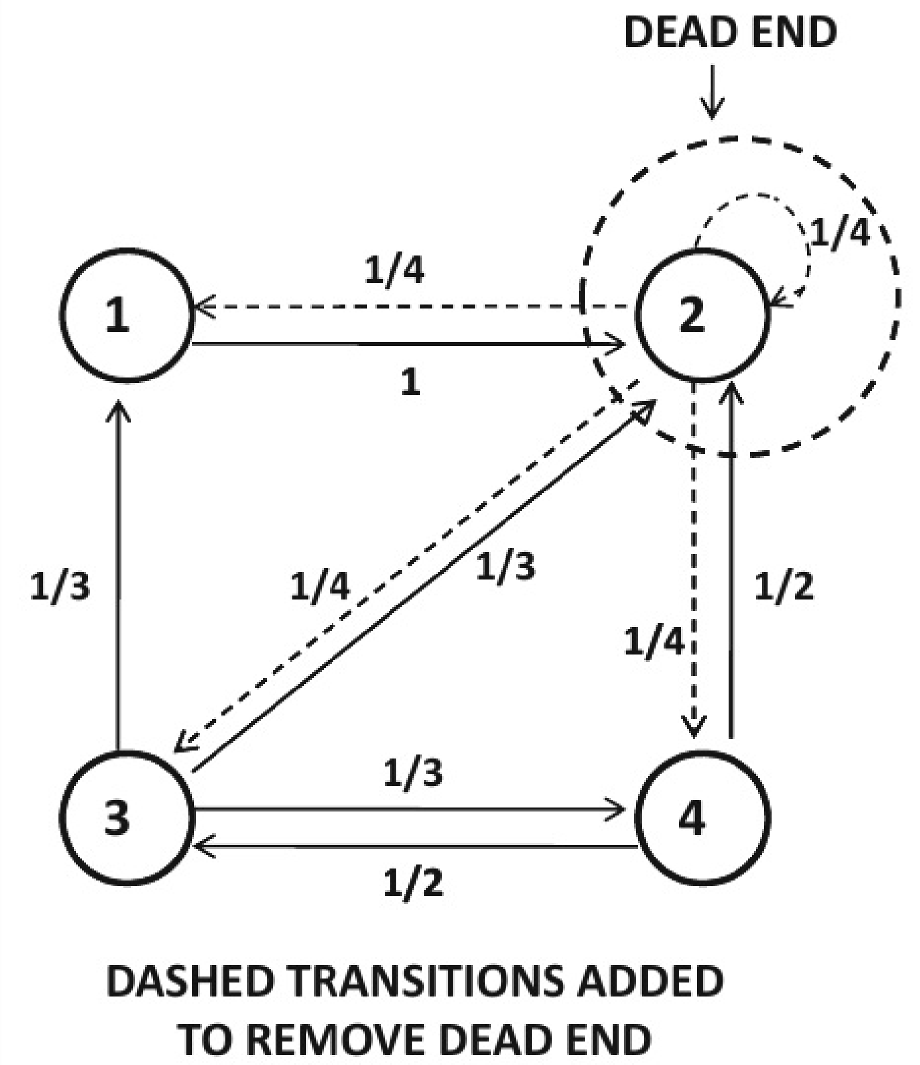


image source Aggarwal Fig. 18.2

Problem: the surfer can get trapped!

Dead-end components don't contain out-going links from the group
⇒ all steady-state probability becomes concentrated into such groups!

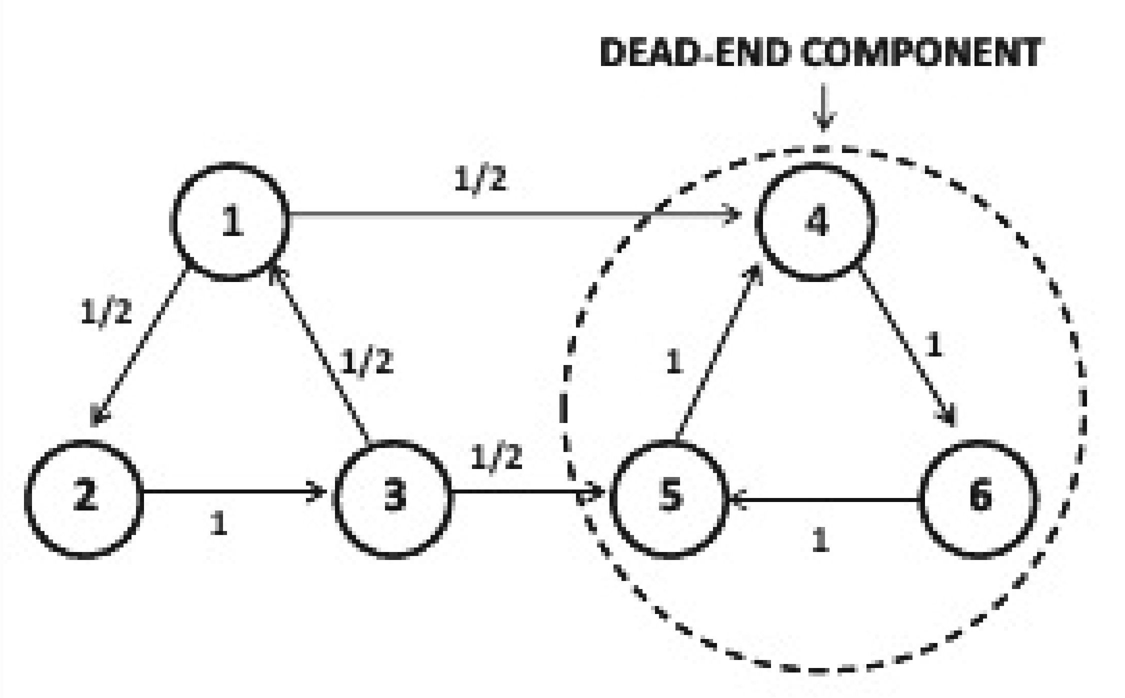


image source Aggarwal Fig. 18.2

Teleportation offers a solution to dead-end components

At each transition, a random surfer may

- jump to an arbitrary page with probability α or
- follow one of links with probability $1 - \alpha$

α = **smoothing** or **damping probability**

- typically $\alpha = 0.1$
- large α makes steady-state probability distribution more uniform
- What happens if $\alpha = 1$?

Presentations as a Markov chain

- graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, $\mathbf{V}$ = pages, $|\mathbf{V}| = n$, $\mathbf{E}$ = links
- $In(v) = \{u \mid (u, v) \in \mathbf{E}\}$ nodes pointing to v
- $Out(v) = \{u \mid (v, u) \in \mathbf{E}\}$ nodes pointed by v
- transition matrix $\mathbf{P}$, where $p_{ij} = \mathbf{P}[i, j]$ probability of transitioning $v_i \rightarrow v_j$
- set $p_{ij} = \frac{1}{|Out(v_i)|}$
- $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)^T$ steady-state probabilities

$$\pi_i = \frac{\alpha}{n} + (1 - \alpha) \sum_{v_j \in In(v_i)} p_{ji} \cdot \pi_j$$

Computation with the power iteration method

Idea: update $\boldsymbol{\pi}$ iteratively, notate by $\boldsymbol{\pi}^{(t)}$ the current $\boldsymbol{\pi}$ at time step t.

Let $\boldsymbol{\alpha} = (\alpha, \alpha, \ldots, \alpha)$ (vector of n αs)

- initialize $\pi_i^{(0)} = 1/n$, $t = 0$
- until $\boldsymbol{\pi}^{(t)}$ converges do

  i) calculate $\boldsymbol{\pi}^{(t+1)}$:

$$\boldsymbol{\pi}^{(t+1)} = \frac{\boldsymbol{\alpha}}{n} + (1 - \alpha)\mathbf{P}^T \boldsymbol{\pi}^{(t)}$$

  ii) normalize such that $\sum \pi_i = 1$

Computation with the power iteration method

Given convergation threshold θ:

- initialize $\pi_i^{(0)} = 1/n$, $t = 0$
- until $(\|\boldsymbol{\pi}^{(t)} - \boldsymbol{\pi}^{(t+1)}\| \leq \theta)$ do
  - for all $i = 1, \ldots, n$

$$\pi_i^{(t+1)} = \frac{\alpha}{n} + (1-\alpha) \sum_{v_j \in In(v_i)} p_{ji} \pi_j^{(t)}$$

  - for all $i = 1, \ldots, n$ normalize $\pi_i^{(t+1)}$s

Note: heavy computation! $\rightarrow$ calculate beforehand for all pages

Alternatively integrate teleportation probabilities into the transition matrix (here $\alpha = 0.1$)

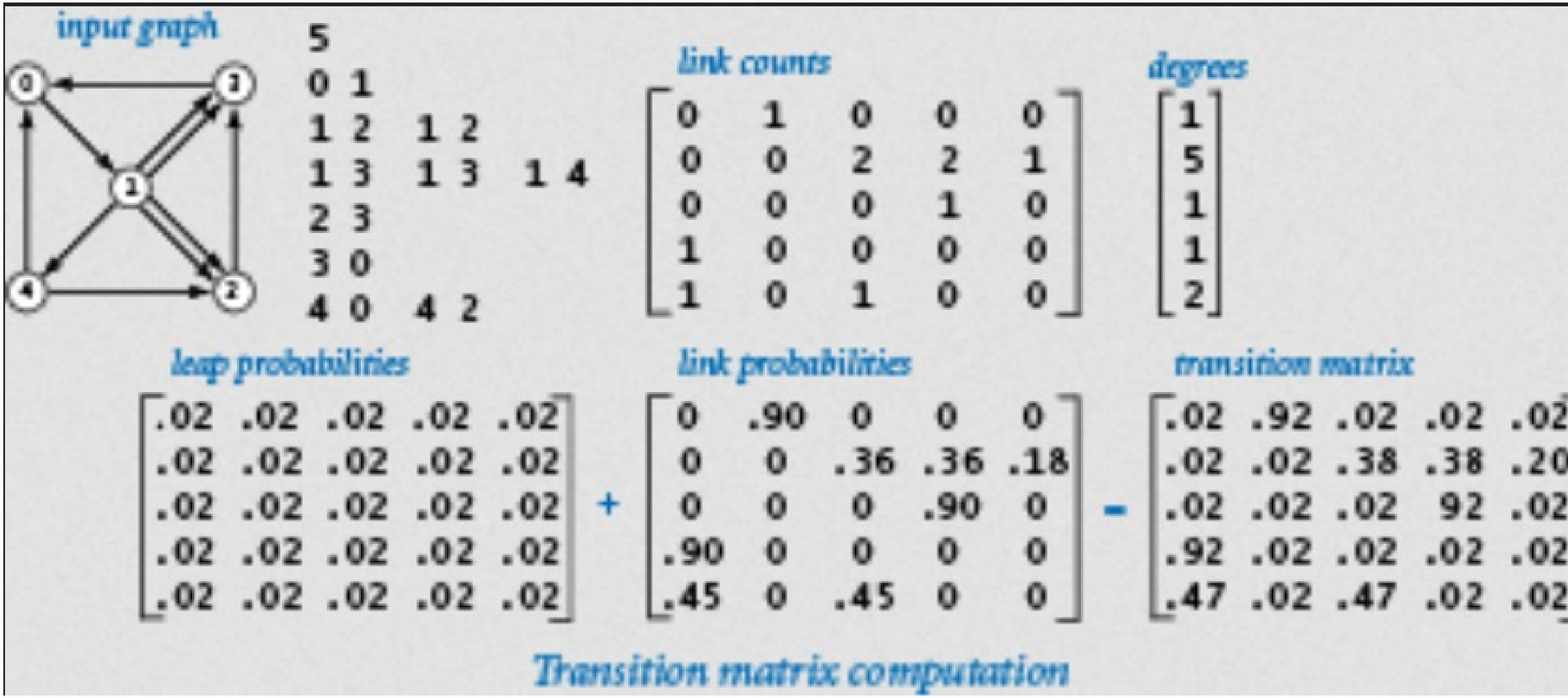


image source
https://introcs.cs.princeton.edu/python/16pagerank/

Other PageRank style methods

1. Topic-sensitive PageRank favours certain topics (like user's interests)

- fix a list of base topics
- search high quality sample of pages from each topic
- restrict teleportation only to these pages

2. SimRank measures similarity between nodes

Intuition: Two random surfers walking backwards from nodes i and j take $L(i, j)$ steps, until they meet. Then SimRank(i, j) is expected value of $c^{L(i,j)}$, where c = decay constant.

SimRank

$$SimRank(v_i, v_j) =$$

$$\begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } |In(v_i)| = 0 \text{ or } |In(v_j)| = 0 \\ \frac{c}{|In(v_i)| \cdot |In(v_j)|} \sum_{p \in In(v_i)} \sum_{q \in In(v_j)} SimRank(p, q) & \text{otherwise} \end{cases}$$

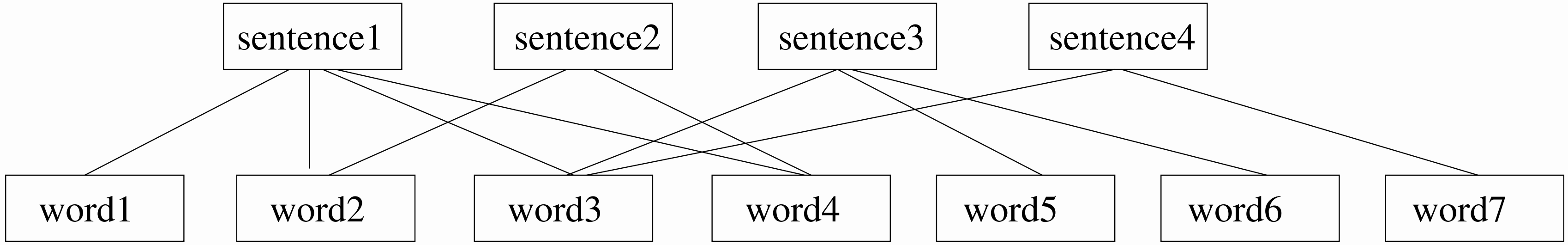
Further reading Aggarwal Ch 18.4.1.2

Summary

- Important problem how to rank documents!
- Content-based scores based on page contents
- Reputation-based scores utilize e.g., link structure
- Ranking algorithms:
  - HITS: query-dependent $\rightarrow$ smaller input graph
  - PageRank: query-independent $\rightarrow$ calculate for all pages (heavy!)

Extra: Applying HITS to sentences and words

Given a keyword or words, create a bipartite graph:



- e.g., all sentences where the given word occurs, all their words, all their sentences and randomly some of their words
- sentences (s) are given S weights and words (w) W weights
- initialize uniformly such that the square sum is 1

Extra: Applying HITS to sentences and words

- update rule: $S(s) = \sum_{w_i \in s} W(w_i)$ and $W(w) = \sum_{w \in s_i} S(s_i)$ + normalization
- in the end the words with largest S-weight are most similar
- analogously search the most similar sentences with a given one
- quality of results varies a lot depending on the documents, language, preprocessing, and how the graph was constructed!