

# CSCI5070 Advanced Topics in Social Computing

## Tutorial 3: Link Analysis & Memory-based Collaborative Filtering

Baichuan Li

The Chinese University of Hong Kong

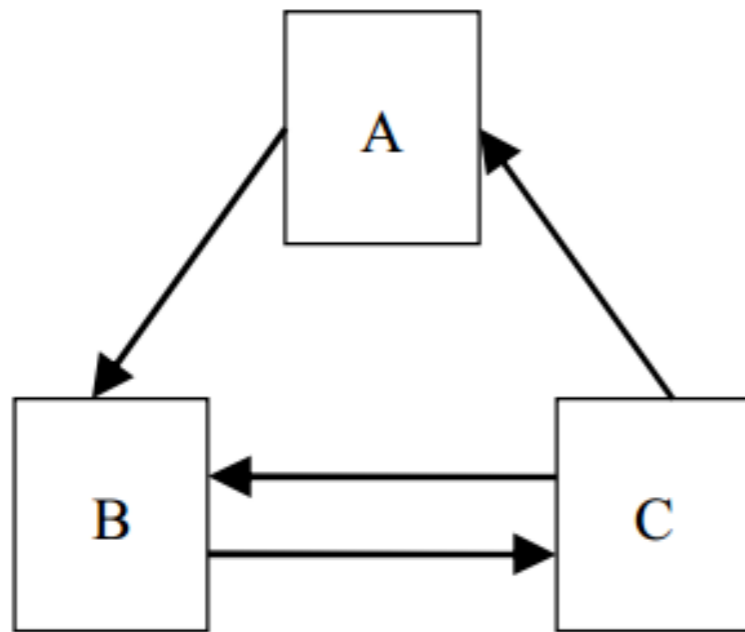
[bcli@cse.cuhk.edu.hk](mailto:bcli@cse.cuhk.edu.hk)

# Outline

- NetworkX
  - PageRank
  - HITS
- Crab
  - Introduction
  - Installing
  - Running a first recommender engine
  - Building a Recommender System

# PageRank

$$PR(A) = \frac{1-d}{N} + d \left( \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \dots \right)$$



The number of web pages  $N = 3$   
The damping parameter  $d = 0.7$

$$\begin{aligned} PR(A) &= (1-d) \times (1/N) + d \times (PR(C)/2) \\ PR(B) &= (1-d) \times (1/N) + d \times (PR(A)/1 + PR(C)/2) \\ PR(C) &= (1-d) \times (1/N) + d \times (PR(B)/1) \end{aligned}$$

So

$$\begin{aligned} PR(A) &= 0.1 + 0.35 \times PR(C) \\ PR(B) &= 0.1 + 0.70 \times PR(A) + 0.35 \times PR(C) \\ PR(C) &= 0.1 + 0.70 \times PR(B) \end{aligned}$$

By solving the above system of linear equations, we get

$$\begin{aligned} PR(A) &= 0.2314 \\ PR(B) &= 0.3933 \\ PR(C) &= 0.3753 \end{aligned}$$

# PageRank in NetworkX

## PageRank

PageRank analysis of graph structure.

<code>pagerank(G[, alpha, personalization, ...])</code>	Return the PageRank of the nodes in the graph.
<code>pagerank_numpy(G[, alpha, personalization, ...])</code>	Return the PageRank of the nodes in the graph.
<code>pagerank_scipy(G[, alpha, personalization, ...])</code>	Return the PageRank of the nodes in the graph.
<code>google_matrix(G[, alpha, personalization, ...])</code>	Return the Google matrix of the graph.

# PageRank in NetworkX

```
pagerank(G, alpha=0.85, personalization=None, max_iter=100, tol=1e-08,  
nstart=None, weight='weight')
```

Return the PageRank of the nodes in the graph.

PageRank computes a ranking of the nodes in the graph  $G$  based on the structure of the incoming links. It was originally designed as an algorithm to rank web pages.

**Parameters :** **G** : graph

A NetworkX graph

**alpha** : float, optional

Damping parameter for PageRank, default=0.85

**personalization** : dict, optional :

The "personalization vector" consisting of a dictionary with a key for every graph node and nonzero personalization value for each node.

**max\_iter** : integer, optional

Maximum number of iterations in power method eigenvalue solver.

**tol** : float, optional

Error tolerance used to check convergence in power method solver.

**nstart** : dictionary, optional

Starting value of PageRank iteration for each node.

**weight** : key, optional

Edge data key to use as weight. If None weights are set to 1.

**Returns :** **pagerank** : dictionary

Dictionary of nodes with PageRank as value

# Example

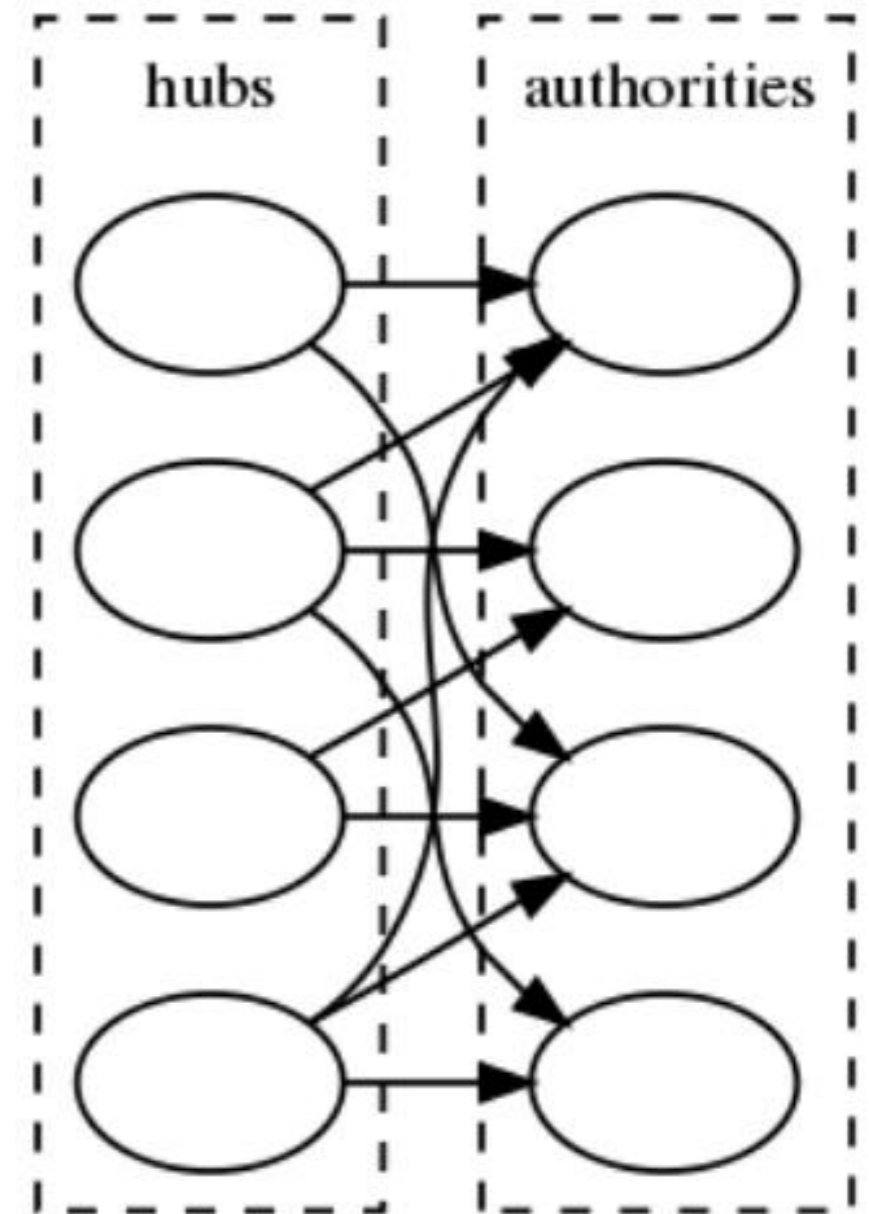
```
import networkx as nx
import matplotlib.pyplot as plt

G=nx.gnp_random_graph(10,0.35) #ER graph
pr=nx.pagerank(G,alpha=0.9)

nx.draw(G)
plt.savefig("pr.png")
```

# HITS

- Hypertext Induced Topics Search (HITS) developed by Jon Kleinberg
- Uses **hubs** and **authorities** to define a recursive relationship between web pages
- An authority is a page that many hubs link to
- A hub is a page that links to many authorities



# HITS in NetworkX

## Hits

Hubs and authorities analysis of graph structure.

[hits](#)(G[, max\_iter, tol, nstart]) Return HITS hubs and authorities values for nodes.

---

[hits\\_numpy](#)(G) Return HITS hubs and authorities values for nodes.

---

[hits\\_scipy](#)(G[, max\_iter, tol]) Return HITS hubs and authorities values for nodes.

---

[hub\\_matrix](#)(G[, nodelist]) Return the HITS hub matrix.

---

[authority\\_matrix](#)(G[, nodelist]) Return the HITS authority matrix.

---



# HITS in NetworkX

```
hits(G, max_iter=100, tol=1e-08, nstart=None)
```

Return HITS hubs and authorities values for nodes.

The HITS algorithm computes two numbers for a node. Authorities estimates the node value based on the incoming links. Hubs estimates the node value based on outgoing links.

**Parameters** G : graph

:

A NetworkX graph

**max\_iter** : interger, optional

Maximum number of iterations in power method.

**tol** : float, optional

Error tolerance used to check convergence in power method iteration.

**nstart** : dictionary, optional

Starting value of each node for power method iteration.

**Returns** : **(hubs,authorities)** : two-tuple of dictionaries

Two dictionaries keyed by node containing the hub and authority values.

# Example

```
import networkx as nx
import matplotlib.pyplot as plt

#Return the GNR digraph with n nodes and redirection probability p.
G=nx.gnr_graph(20,0.15)

h,a=nx.hits(G)

nx.draw(G)
plt.savefig("HITS.png")
```

# Crab

- Crab is a Python framework for building recommender engines integrated with the world of scientific Python packages ([numpy](#), [scipy](#), [matplotlib](#))
- Features:
  - **Recommender Algorithms:** User-Based Filtering and Item-Based Filtering
  - **Work in progress:** Slope One, SVD, Evaluation of Recommenders
  - **Planned:** Sparse Matrices, REST API's



# Installing

- Easy install
  - `pip install -U crab`
  - `easy_install -U crab`
- Install from source package
  - Download the package from <https://github.com/muricoca/crab/downloads>
  - `python setup.py install`

# Running A First Recommender Engine

- Creating the input
  - Consists of an user ID and an item ID
  - Numbers expressing the strength of the user's preference for the items
    - Generally larger values mean stronger positive preferences
    - E.g., if those values are ratings on a scale 1 to 5, the 1 could mean items that the user can't stand, and 5 as favorite items
- Crab includes a few standard datasets

```
>>> from scikits.crab import datasets
>>> movies = datasets.load_sample_movies()
>>> songs = datasets.load_sample_songs()
```

# Running A First Recommender Engine

- Creating the input
  - Consists of an user ID and an item ID
  - Numbers expressing the strength of the user's preference for the items
    - Generally larger values mean stronger positive preferences
    - E.g., if those values are ratings on a scale 1 to 5, the 1 could mean items that the user can't stand, and 5 as favorite items
- Crab includes a few standard datasets

```
>>> from scikits.crab import datasets
>>> movies = datasets.load_sample_movies()
>>> songs = datasets.load_sample_songs()
```

# Example

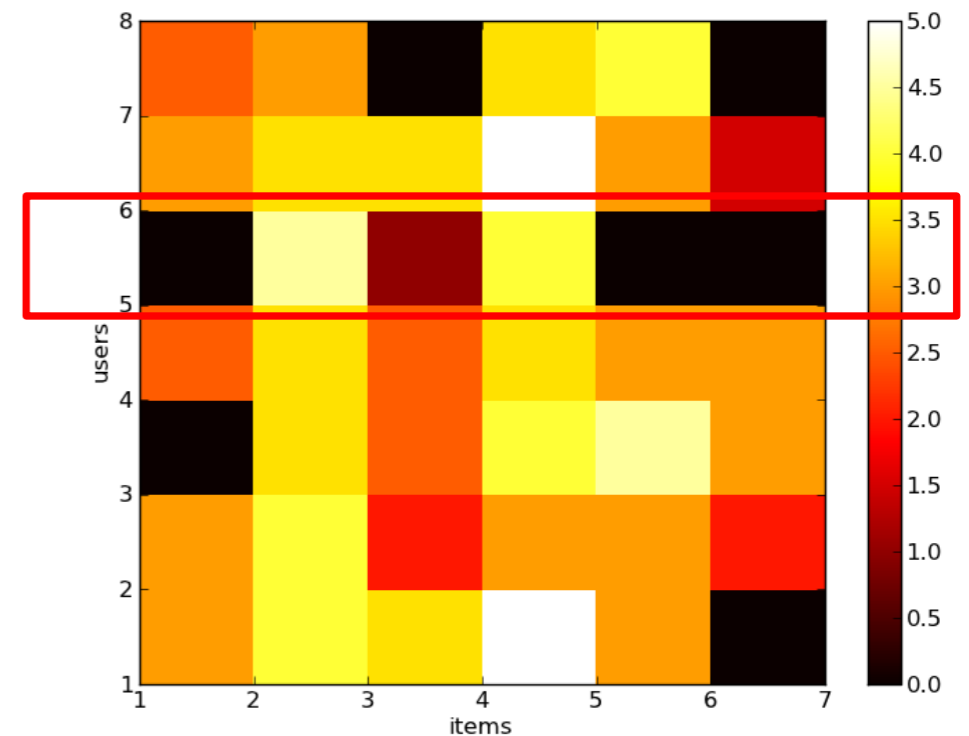
- Format
- {user\_id:{item\_id: preference, item\_id2: preference, ...}, user\_id2: {...}, ...}

```
>>> print movies.data
{1: {1: 3.0, 2: 4.0, 3: 3.5, 4: 5.0, 5: 3.0},
 2: {1: 3.0, 2: 4.0, 3: 2.0, 4: 3.0, 5: 3.0, 6: 2.0},
 3: {2: 3.5, 3: 2.5, 4: 4.0, 5: 4.5, 6: 3.0},
 4: {1: 2.5, 2: 3.5, 3: 2.5, 4: 3.5, 5: 3.0, 6: 3.0},
 5: {2: 4.5, 3: 1.0, 4: 4.0},
 6: {1: 3.0, 2: 3.5, 3: 3.5, 4: 5.0, 5: 3.0, 6: 1.5},
 7: {1: 2.5, 2: 3.0, 4: 3.5, 5: 4.0}}
```

```
>>> print movies.user_ids
{1: 'Jack Matthews',
 2: 'Mick LaSalle',
 3: 'Claudia Puig',
 4: 'Lisa Rose',
 5: 'Toby',
 6: 'Gene Seymour',
 7: 'Michael Phillips'}
>>>
>>> print movies.item_ids
{1: 'Lady in the Water',
 2: 'Snakes on a Planet',
 3: 'You, Me and Dupree',
 4: 'Superman Returns',
 5: 'The Night Listener',
 6: 'Just My Luck'}
```

# Building a Recommender System

- Goal: To recommend a movie to Toby (user 5)
- Toby already watched these movies:
  - Snakes on a Planet (item 2)
  - You, Me and Dupree (item 3)
  - Superman Returns (item 4)
- Recommendation is typically about discovering new things
- Estimate Toby's "ratings" on item 1, 5, and 6





# Code

```
>>> from scikits.crab.models import MatrixPreferenceDataModel
>>> #Build the model
>>> model = MatrixPreferenceDataModel(movies.data)
>>>
>>> from scikits.crab.metrics import pearson_correlation
>>> from scikits.crab.similarities import UserSimilarity
>>> #Build the similarity
>>> similarity = UserSimilarity(model, pearson_correlation)
>>>
>>> from crab.recommenders.knn import UserBasedRecommender
>>> #Build the User based recommender
>>> recommender = UserBasedRecommender(model, similarity, with_preference=True)
>>> #Recommend items for the user 5 (Toby)
>>> recommender.recommend(5)
[(5, 3.3477895267131013), (1, 2.8572508984333034), (6, 2.4473604699719846)]
```

# Evaluation

- Training and Testing Data
- Evaluation metrics
  - Root-mean-square error (RMSE): the square root of the average of the squares of the differences between actual and estimated preference values
  - Mean absolute error (MAE): the average absolute value of the differences between actual and estimated preference values
  - The lower, the better!

# References

- <http://muricoca.github.com/crab/tutorial.html>
- <http://networkx.lanl.gov/contents.html>
- [http://www.cis.hut.fi/Opinnot/T-61.6020/2008/pagerank\\_hits.pdf](http://www.cis.hut.fi/Opinnot/T-61.6020/2008/pagerank_hits.pdf)
- [http://www.dcs.bbk.ac.uk/~dell/teaching/ir/examples/pr\\_example.pdf](http://www.dcs.bbk.ac.uk/~dell/teaching/ir/examples/pr_example.pdf)
- [http://en.wikipedia.org/wiki/HITS\\_algorithm](http://en.wikipedia.org/wiki/HITS_algorithm)