NYU Center for Data Science

Week 11.1: PageRank

DS-GA 1004: Big Data

This week

- PageRank
  - [Page, Brin, Motwani, Winograd, 1999]
- Extensions to PageRank

Web search

- Early search engines relied on matching text in **query** to text in **web page**

- Pages were crawled (spidered) at regular intervals and added to an index
- We've already seen some tools for indexing and searching
- ***What can go wrong with this approach?***

Yahoo.com in April, 1997 [archive.org]

Spam attacks

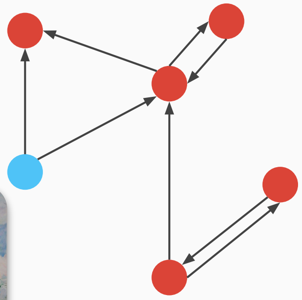
- Imagine that you want to get lots of traffic to your website
- You know that it is indexed regularly by search engines
- **Idea**: fill up your page with all the **most popular search terms**
- End result: **$$$** selling dinosaur memorabilia

PageRank: use the network!

- Key insight: the structure of the web has information content!
- Publishers are more likely to link to pages that they trust
- It’s **easy** to make a spammy page
- It’s **hard** to get other people to link to it

The random surfer model

- Imagine the web as a directed graph
  - Nodes = pages
  - Edges = links
- Model a user's activity as a random walk
  - **P**[Going to page *v* | Currently at page *u*] = [$u \rightarrow v$] / out-degree(*u*)
- Users are more likely to land at pages with **high in-degree**
  - What is the steady-state distribution **P[v]**?



Markov chains

- Let **M**[v, u] = P[v | u]
    - Columns (u) = current states (N pages) Rows (v) = next states (N pages)

- **M** is a **stochastic matrix**: non-negative, each column sums to 1

Markov chains

- Let **M**[v, u] = P[v | u]
  - Columns (u) = current states (N pages)   Rows (v) = next states (N pages)
- **M** is a **stochastic matrix**: non-negative, each column sums to 1
- Let **p** be a non-negative vector in $\mathbf{R}^N$ that sums to 1
  - (**Mp**)[v] marginalizes over *u* to compute probability of state *v*

Markov chains

- Let **M**[v, u] = P[v | u]
  - Columns (u) = current states (N pages)   Rows (v) = next states (N pages)
- **M** is a **stochastic matrix**: non-negative, each column sums to 1
- Let **p** be a non-negative vector in $\mathbf{R}^N$ that sums to 1
  - (**M**p)[v] marginalizes over *u* to compute probability of state *v*
- One step maps

$$p[v] \longrightarrow (\mathbf{M}p)[v] = \sum_u P[v \mid u] * p[u] = p'[v]$$

Steady-state distributions

- A steady-state distribution satisfies **p** = **Mp**

- Such a **p** exists if there is a *path* connecting every pair u → v
  - We say that **M** is ***irreducible*** or ***strongly connected***
  - (This is a **sufficient**, but **not necessary** condition!)

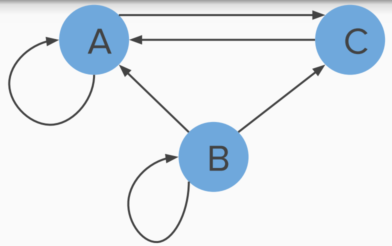
Steady-state distributions

- A steady-state distribution satisfies **p** = **Mp**
- Such a **p** exists if there is a *path* connecting every pair u → v
  - We say that **M** is ***irreducible*** or ***strongly connected***
  - (This is a **sufficient**, but **not necessary** condition!)
- **p** ∝ **eigenvector** of **M** with **eigenvalue** 1
  - The largest possible for a stochastic matrix!
- PageRank(*u*) = p[*u*] = probability of random surfer being at node *u*

Markov chain Eigenvectors

- Standard eigenvector solvers do not scale to the web
  - $O(N^3)$ cost to solve in general, can be smaller if sparse, but N can still be huge!
- Instead, use **power iteration**
  - Initialize $p_0[u] \leftarrow 1/N$ (uniform distribution)
  - For $i = 1, 2, \ldots, T_{max}$
    - $p_i \leftarrow \mathbf{M} p_{i-1}$ ($p_i = \mathbf{M}^i p_0$) ← Parallelize here over rows of M
- This will eventually converge to the stationary distribution
  - (If such a distribution exists…)

Example



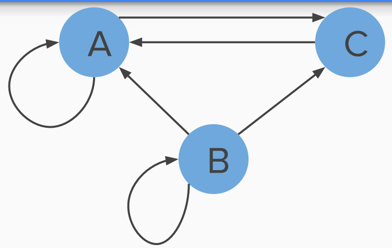
p_1
?
?
?

	A	B	C
A	½	⅓	1
B	0	⅓	0
C	½	⅓	0

p_0
⅓
⅓
⅓

- Initialize $p_0[u] \leftarrow 1/N$
- For i = 1, 2, …, T_{max}
  - $p_i \leftarrow \mathbf{M}p_{i-1}$

Example



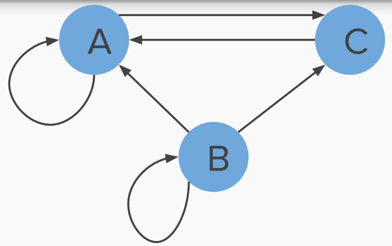
p_1
11/18
2/18
5/18

	A	B	C
A	½	⅓	1
B	0	⅓	0
C	½	⅓	0

p_0
⅓
⅓
⅓

- Initialize $p_0[u] \leftarrow 1/N$
- For i = 1, 2, …, T_{max}
  - $p_i \leftarrow \mathbf{M}p_{i-1}$

Example



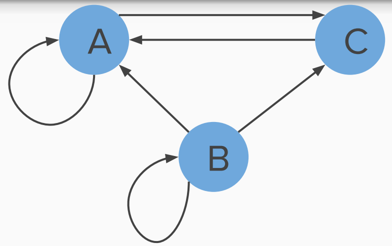
p_2
67/108
4/108
37/108

	A	B	C
A	½	⅓	1
B	0	⅓	0
C	½	⅓	0

p_1
11/18
2/18
5/18

- Initialize $p_0[u] \leftarrow 1/N$
- For $i = 1, 2, \ldots, T_{max}$
  - $p_i \leftarrow \mathbf{M}p_{i-1}$

Example



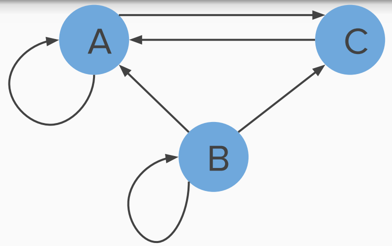
p_3
431/648
8/648
209/648

	A	B	C
A	½	⅓	1
B	0	⅓	0
C	½	⅓	0

p_2
67/108
4/108
37/108

- Initialize $p_0[u] \leftarrow 1/N$
- For $i = 1, 2, \ldots, T_{max}$
  - $p_i \leftarrow \mathbf{M}p_{i-1}$

Example



p_n
2/3
0
1/3

	A	B	C
A	½	⅓	1
B	0	⅓	0
C	½	⅓	0

p_n
2/3
0
1/3

- Initialize $p_0[u] \leftarrow 1/N$
- For $i = 1, 2, \ldots, T_{max}$
    - $p_i \leftarrow \mathbf{M}p_{i-1}$

Checking your work in Python

- M[v, u] = P[v | u] (probability of going to v from u)
    - Each column is a probability distribution
    - ⇒ **M.sum(axis=0)** should be all ones
- **evals, evecs = np.linalg.eig(M)** is almost, but not quite, what we want

Checking your work in Python

- **M**[v, u] = P[v | u] (probability of going to v from u)
  - Each column is a probability distribution
  - ⇒ **M.sum(axis=0)** should be all ones

- **evals, evecs = np.linalg.eig(M)** is almost, but not quite, what we want

- **Remember**:
  Eigenvectors have unit Euclidean (L_2) norm: $\boldsymbol{v} = \boldsymbol{v} \,/\, \|\boldsymbol{v}\|_2$
  Distributions have unit L_1 norm: $\boldsymbol{p} = \boldsymbol{v} \,/\, \|\boldsymbol{v}\|_1 \neq \boldsymbol{v}$

Checking your work in Python

- M[v, u] = P[v | u] (probability of going to v from u)
  - Each column is a probability distribution
  - ⇒ **M.sum(axis=0)** should be all ones

- **evals, evecs = np.linalg.eig(M)** is almost, but not quite, what we want

- **Remember**:

  Eigenvectors have unit Euclidean (L_2) norm: $\mathbf{v} = \mathbf{v} \,/\, \|\mathbf{v}\|_2$

  Distributions have unit L_1 norm: $\mathbf{p} = \mathbf{v} \,/\, \|\mathbf{v}\|_1 \neq \mathbf{v}$

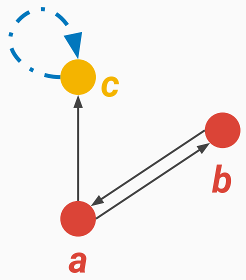
- Re-normalize each column:
  - **evecs /= np.abs(evecs).sum(axis=0, keepdims=True)**

You may also have to flip the sign of *v*.

If *v* is an eigenvector, so is -*v*!

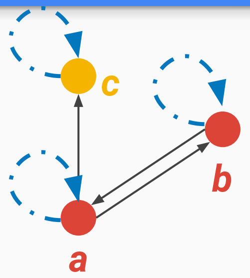
What about sinks?

- If you have a **vertex with no outgoing edges**, its column in M is all zeros.
  - This is not a valid probability distribution!
- Common fix: add a **self-loop** to any nodes with out-degree 0
  - **Nodes with outgoing edges** do not need to be modified
- Result is a well-formed transition matrix



Spider traps

- Node ***c*** is a sink in this graph, also known as a **spider trap**
- A random surfer landing at ***c*** can never leave!



	a	b	c
a	⅓	½	0
b	⅓	½	0
c	⅓	0	1

p_0
⅓
⅓
⅓

Power iteration

p
0
0
1

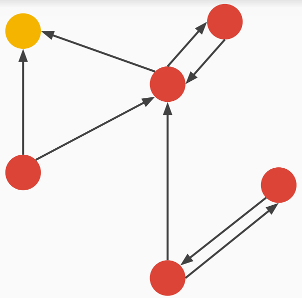
Traps are easy to create!

All it takes is one link from a well-connected vertex to cause serious damage!

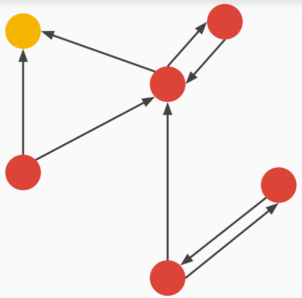
Teleportation / random restart

- The web is decidedly **not strongly connected**

- Not all pages have outward links (column sum = 0)



Teleportation / random restart



- The web is decidedly **not strongly connected**

- Not all pages have outward links (column sum = 0)

- Solution: **teleportation**!

$$M[v, u] \rightarrow a * M[v, u] + (1 - a) * 1/N$$

- With probability a, follow the links
  With probability 1-a, jump uniformly at random

Using PageRank in search

- PageRank scores each vertex (page) according to network topology
    - p[v] > p[u] ⇒ v is “better connected” than u
    - It does not use content! But still a reasonable measure of importance
- Basic search implementation:
    - Use **text search** (LSH, etc) to find candidate items
    - Use **pagerank** (and other cues) to order results
- Can we do better?

… find out in part 2!