

Numerical Linear Algebra – PageRank continued

CS370 Lecture 27 – March 17, 2017
St. Patrick's Day edition



Last Time - PageRank



In the last lecture we...

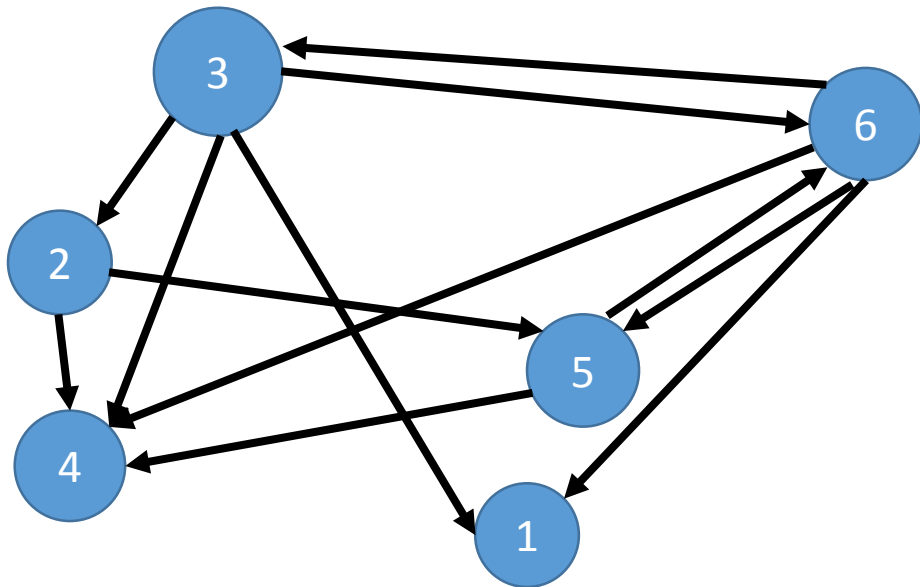
- introduced the simple “random surfer” model for ranking web pages
- began describing the random surfing process with a “Google matrix” of transition *probabilities*.

Today:

- work through a specific Google matrix example
- look at its properties
- explore how it’s used in the PageRank algorithm

Example Problem:

Construct the google matrix $M = \alpha \left(P + \frac{1}{R} e d^T \right) + (1 - \alpha) \frac{1}{R} e e^T$ for the small web shown here, using $\alpha = 1/2$, and $R = 6$ pages.



Recall:

$$P_{ij} = \begin{cases} \frac{1}{\deg(j)}, & \text{if link } j \rightarrow i \text{ exists} \\ 0, & \text{otherwise} \end{cases}$$

$$d_i = \begin{cases} 1, & \text{if } \deg(i) = 0 \\ 0, & \text{otherwise} \end{cases}$$

$$e = [1, 1, 1, \dots, 1]^T$$

Some Useful Properties of M

The entries of M satisfy $0 \leq M_{ij} \leq 1$.

Probabilities outside this range are essentially meaningless here.

Each column of M sums to 1.

$$\sum_{i=1}^R M_{ij} = 1$$

Interpretation: if we are currently on a webpage, probability of being on ***some*** webpage after a transition is 1. (i.e. we don't just disappear).

Markov Transition Matrices

The google matrix M is an example of a *Markov matrix*.

We define a Markov matrix Q by the two properties we just saw:

$$0 \leq Q_{ij} \leq 1$$

and

$$\sum_i Q_{ij} = 1$$

Probability Vector

Now, define a **probability vector** as a vector q such that

$$0 \leq q_i \leq 1$$

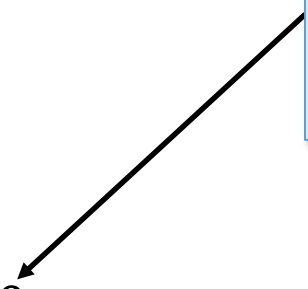
and

$$\sum_i q_i = 1.$$

If we start our surfer at a random page with equal probabilities, this can be represented by a probability vector, with $p_i = \frac{1}{R}$.

Evolving The Probability Vector

Superscripts
indicate number
of transitions
taken.



Now we have:

- the probability vector describing the **initial state**, p^0 .
- a Markov matrix M describing the **transition probabilities** among pages.

Their *product* Mp^0 tells us the probabilities of our surfer being at each page after **one transition**.

$$p^1 = Mp^0$$

Likewise, for any step n , next step probabilities are, $p^{n+1} = Mp^n$.

Evolving The Probability Vector: Example #1

e.g., $p^0 = [1, 0, 0, 0]^T$. (We're *definitely* on page 1.)

If we had a google/transition matrix $M = \begin{bmatrix} 1/3 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 0 \\ 1/3 & 1/2 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \end{bmatrix}$,

then after one step, what is $p^1 = Mp^0$? And what does it mean?

$$p^1 = \left[\frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3} \right]^T$$

~33% chance of being at page 1, 3, or 4 after one step, starting from page 1.

Evolving The Probability Vector: Example #2

e.g., $p^0 = \left[\frac{1}{2}, 0, \frac{1}{2}, 0\right]^T$. (We're on page 1 or 3 with probability 0.5 each.)

If we have same matrix $M = \begin{bmatrix} 1/3 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 0 \\ 1/3 & 1/2 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \end{bmatrix}$,

then after one step, what is p^1 ?

Preserving a Probability Vector

If p^n is a probability vector, is $p^{n+1} = Mp^n$ necessarily one also?

(ie. Do we have: $0 \leq p_i^{n+1} \leq 1$ and $\sum_i p_i^{n+1} = 1$?)

Yes! First, why non-negative?

We have $p_i^{n+1} \geq 0$, since it is just sums & products of probabilities ≥ 0 .

Preserving a Probability Vector

We can also show $\sum_i p_i^{n+1} = 1$, as follows:

$$\sum_i p_i^{n+1} = \sum_i \sum_j M_{ij} p_j^n = \sum_j p_j^n \sum_i M_{ij} = \sum_j p_j^n = 1$$

By def'n of mat-
vec multiply

Reordering
summation

$\text{colsum}(M)=1$

Since p^n was a
probability vector.

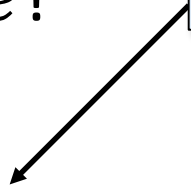
To be a probability vector, also need $p_i^{n+1} \leq 1$? Why is this true?

Page Rank idea

Finally, Page Rank asks:

With what **probability** does our surfer end up at each page after **many** steps, starting from $p^0 = \frac{1}{R} e$?

Exponent, (not step index)



i.e., What is $p^\infty = \lim_{k \rightarrow \infty} (M)^k p^0$?

Higher probability in p^∞ implies greater importance.

Then we can rank the pages by this importance measure.

Page Rank algorithm summary

1. Given a graph of a network, compute a corresponding Google transition (Markov) matrix...

$$M = \alpha \left(P + \frac{1}{R} e d^T \right) + (1 - \alpha) \frac{1}{R} e e^T$$

2. Repeatedly evolve a probability vector p^i via $p^{n+1} = M p^n$ towards a steady state, approximating a “random surfer”.
3. The site with the highest probability of being visited is considered most important/influential.

Page Rank Example - Results

Starting from $p^0 = \frac{1}{R} e$, repeated multiplication by M gives a sequence of probability vectors, eventually settling down.

					Page #	Ranking
$\begin{bmatrix} 0.16667 \\ 0.16667 \\ 0.16667 \\ 0.16667 \\ 0.16667 \\ 0.16667 \end{bmatrix}$,	$\begin{bmatrix} 0.09583 \\ 0.16666 \\ 0.11944 \\ 0.26111 \\ 0.16666 \\ 0.19027 \end{bmatrix}$,	$\begin{bmatrix} 0.08245 \\ 0.12318 \\ 0.08934 \\ 0.28118 \\ 0.19342 \\ 0.23041 \end{bmatrix}$		
p^0		Mp^0		M^2p^0		
			,	$\begin{bmatrix} 0.06776 \\ 0.10280 \\ 0.07749 \\ 0.32051 \\ 0.18726 \\ 0.24415 \end{bmatrix}$		
				M^3p^0		
				\dots		
				$\begin{bmatrix} 0.05205 \\ 0.07428 \\ 0.05782 \\ 0.34797 \\ 0.19975 \\ 0.26810 \end{bmatrix}$		
				$M^{10}p^0$		
					1	6
					2	4
					3	5
					4	1
					5	3
					6	2

For earlier example, the pages are ranked as: 4, 6, 5, 2, 3, 1 based on these final probabilities.

Questions to Ponder...

- Do we actually know if it will settle (*converge*) to a fixed final result?
- If yes, then how long will it take? Roughly how many *iterations* are needed before we can stop?
- Can we implement this *efficiently* (e.g. for very large networks?)

Making Page Rank Efficient



A naïve implementation of Page Rank involves repeated multiplying massive matrices with $> 1\text{billion} \times 1\text{billion}$ entries, for every search!

How can we implement this in a way that is actually computational feasible?

First step: Precomputation

The ranking vector p^∞ can be pre-computed once and stored, ***independent of any specific query.***

e.g., to search for “lobster hats”, Google finds **only** the subset of pages matching these keywords, and ranks those by their values in the (**precomputed**) p^∞ .



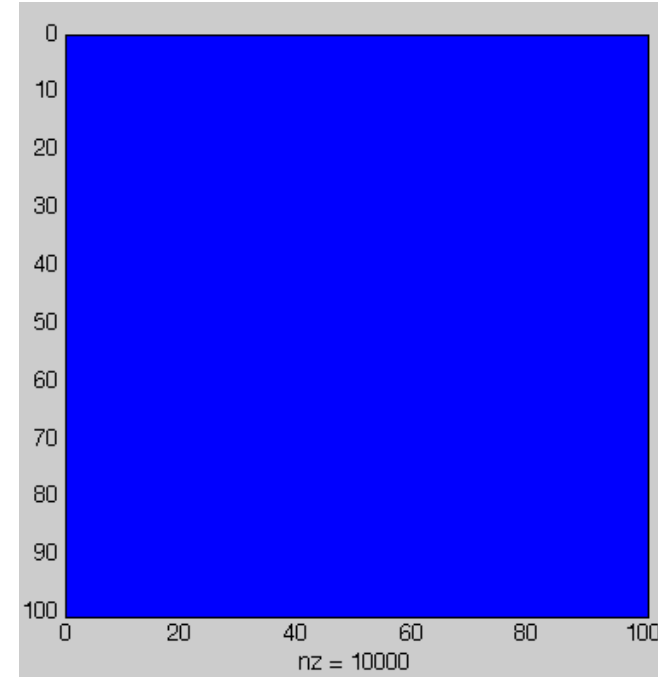
Matrix Sparsity

In numerical linear algebra, we often deal with two kinds of matrices.

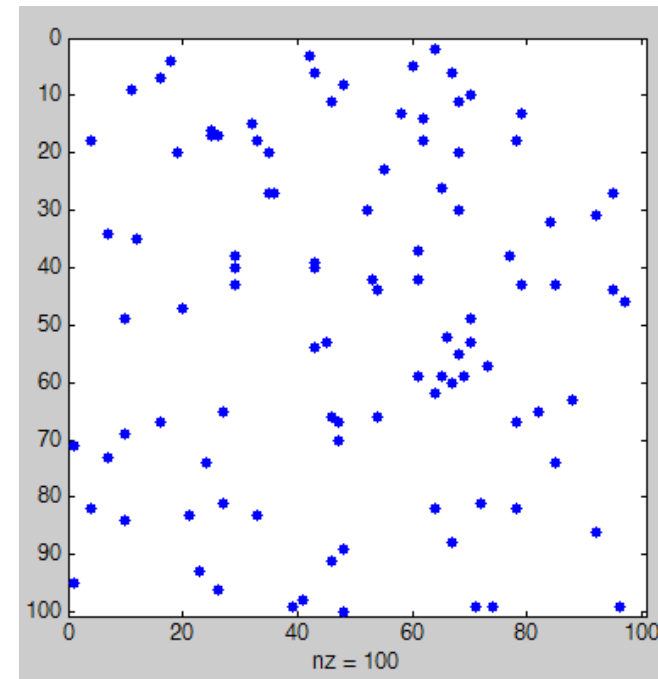
Dense: Most or all entries are **non-zero**. Store in an $N \times N$ array, manipulate “normally”.

Sparse: Most entries are **zero**. Use a “sparse” data structure to save space (and time).

Prefer algorithms that avoid “destroying” sparsity (i.e., filling in zero entries).



Non-zeros
in a dense
matrix



Non-zeros
in a sparse
matrix

e.g. A Simple Sparse Matrix Data Structure

Store three arrays, one entry per non-zero matrix entry:

i: integer row indices

j: integer column indices

value: floating point number

Simple, but not easy to work with.

Many more efficient representations exist, sometimes depending on the specific layout of non-zeroes.

Second Step: Exploiting Sparsity

To implement Page Rank efficiently, it is crucial to **exploit sparsity**.

Sadly, our M matrix was **fully dense**. No zero entries at all!

A dense matrix-vector multiply with 1000000000^2 entries is sloooooow.

The trick: Use linear algebra manipulations to perform the main iteration

$$\mathbf{p}^{n+1} = M\mathbf{p}^n$$

without ever creating/storing M !

Let's derive
why, next time!