

NEU AI fall 2018 / M. Gaidis  
Pre-Course Assignment 0

Python for PageRank

# What is PageRank?

# From Wikipedia (<https://en.wikipedia.org/wiki/PageRank>):

**PageRank** is an algorithm once used by Google Search to rank websites in their search engine results. It is a way of measuring the importance of website pages. [PageRank was named after Larry Page, one of the founders of Google, and helped make him one of the richest men in the world!]

# PageRank works by evaluating the *number* and *quality* of links to a page to determine a rough estimate of how important the website is. One assumes that more important websites are likely to receive more links from other websites. A page that is linked to by many pages (especially pages with high PageRank) receives a high rank itself. The PageRank algorithm outputs a probability that a person randomly clicking on links will arrive at a particular page. The probability is expressed as a different numeric value between 0 and 1 for each website. A PageRank of 0.5 means there is a 50% chance that a person clicking on a random link will be directed to the website with the 0.5 PageRank.

Unfortunately, web site designers have found ways to manipulate their PageRank, and so Google now relies heavily on other methods of generating search results. However, the mathematics of PageRank is still relevant, as it can apply to any graph or network in any domain. Consider some of the following applications:

1. Systems analysis of road networks linking important cities
2. In neuroscience, PageRank correlates with the relative firing rate of a neuron
3. Twitter uses a version of PageRank to recommend other accounts you may wish to follow
4. Scientific impact factor of publications can be approximated with PageRank
5. PageRank can help determine the best academic doctoral programs at different universities
6. PageRank can process language to interpret the likely meaning of a sentence
7. In biology, PageRank can tell you which species are essential to the health of the environment
8. PageRank may improve voting systems so the best candidate is elected

# Your Assignment:

Use simple Python code to create an iterative version of PageRank, and demonstrate it works for a small network that you will create.

# Why is this Assignment Important?

There are two main purposes to this assignment:

1. It shows you how even simple algorithms can give computing machines the power to add to human intelligence on a scale that human brains cannot manage. This algorithm by itself is not “artificial intelligence,” but it is one of many tools used together to create AI. During this class, you will learn about many of these tools and how they contribute to AI. No one tool is “AI,” just as a single hammer does not make someone a carpenter. AI needs many different tools, just like a carpenter needs many tools before they can even become a carpenter. Then, AI needs to figure out what to do with all of these tools to make something useful.
2. It gives you an introduction to Python programming, showing you some of the most important data structures and methods that you need to be familiar with for this class.

# Simple Example of Page Rank Computation

The PageRank of a node is defined *iteratively* and depends on the number and PageRank metric of all nodes that link to it ("incoming links"). The iterative calculation proceeds as follows:

1. Start at iteration i=0 by giving all N nodes in your network an equal PageRank of 1/N
2. Compute PageRank at iteration i=1 using the values from the previous iteration step i=0
3. Continue in this way, computing PageRank at iteration i=k from the values at i=(k-1)
4. Stop iterating when the PageRank values do not change significantly (they converge).

The iteration should not take too long, as Google has revealed that a network with about 300 million links will converge in only 50 iterations. Your networks will be much smaller, and should converge very quickly.

To explain the computation (step 2 above), it is easiest to use an example. Figure 1 shows a network of 5 nodes (A, B, C, D, E) in a directed graph. It is a “directed” graph because the edges that represent links between nodes have a direction specified by an arrowhead. For example, you can imagine a webpage C that has an outlink on it to webpage B. In addition, webpages B and D have outlinks to webpage C – or we can say that C has two inlinks.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Figure 1: Example graph with 5 nodes and multiple directed links | Step 1, iteration i=0  Assign a value of 1/N PageRank (PR) to all nodes in the network.   |  |  | | --- | --- | | Node | PR Value @ i=0 (PR\_0) | | A | 1/5 | | B | 1/5 | | C | 1/5 | | D | 1/5 | | E | 1/5 | |
| Figure 2: Node B shares its 1/5 PageRank equally with all of its outlinks (nodes it links to), shown in red. B has 2 outlinks, so we multiply the previous iteration PageRank by ½, and give that amount to B’s outlinks C and D. Node A has two inlinks (shown in black), and its PR is the sum of 1/3 of D’s PR (1/3 because D has 3 outlinks) plus the entire PR of node E (because E only has 1 outlink). | Step 2, iteration i=1  Compute new PageRank (PR\_1) by adding together all fractional page ranks of inlinks.   |  |  |  | | --- | --- | --- | | n | i=0 | PR\_1 | | A | 1/5 | 1/3\*D + E = 1/3\*1/5 + 1/5 = 0.2667 | | B | 1/5 | A + C = 1/5 + 1/5 = 0.4 | | C | 1/5 | 1/2\*B+1/3\*D = 1/10+1/15 = 0.1667 | | D | 1/5 | 1/2\*B = 1/2\*1/5 = 1/10 = 0.1 | | E | 1/5 | 1/3\*D = 1/3\*1/5 = 1/15 = 0.0667 | |
| Figure 3: Continue to iterate over the nodes, with the new values of PageRank from step 2. | Step 3, iteration i=2  Compute new PageRank by adding together values computed from the previous iteration’s PR.   |  |  |  |  | | --- | --- | --- | --- | | n | i=0 | i=1 | PR\_2 | | A | 0.2 | 0.2667 | 1/3\*D + E = 0.1 | | B | 0.2 | 0.4 | A + C = 0.4334 | | C | 0.2 | 0.1667 | 1/2\*B+1/3\*D = 0.333 | | D | 0.2 | 0.1 | 1/2\*B = 0.2 | | E | 0.2 | 0.0667 | 1/3\*D = 0.0333 | |
| Figure 4: Continue to iterate over the nodes, with the new values of PageRank from previous iteration. | Step 4, iteration converges  Continue repeating step 3 until the values stop changing (within your margin of error).   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | n | i=0 | i=1 | i=2 | i=10 | i=∞ | | A | 0.2 | 0.2667 | 0.1 | 0.1249 | 0.125 | | B | 0.2 | 0.4 | 0.4334 | 0.3752 | 0.375 | | C | 0.2 | 0.1667 | 0.333 | 0.2499 | 0.25 | | D | 0.2 | 0.1 | 0.2 | 0.187 | 0.1875 | | E | 0.2 | 0.0667 | 0.0333 | 0.0629 | 0.0625 |   Note how the columns always sum to 1. |

This is quite straightforward, and should be relatively simple to code up in your favorite programming language. You now have an estimate of the probability that a random web surfer will be at any one of these webpage nodes. The mathematical expression used to describe the example calculations in the previous table is given here:

where is the PageRank of a page , and as shown is dependent on the PageRank values of each page contained in the set (the set containing all pages that have inlinks to page ), and divided by the number of outlinks from page .

There is one significant problem, however, that is unrelated to web designers trying to take advantage of the system. Figure 5 shows addition of two “sink” nodes: F and G. Although there are links to F and G, there are no links back to the initial grouping of 5 nodes. If a page has no links to other pages, it becomes a “sink” and it terminates the random surfing process. Since PageRank is intended to represent a probability that the user will click through to a given node, computing PageRanks for the graph in Figure 5 would suggest that the user would be at node F with 0.5 probability, or at node G with 0.5 probability.

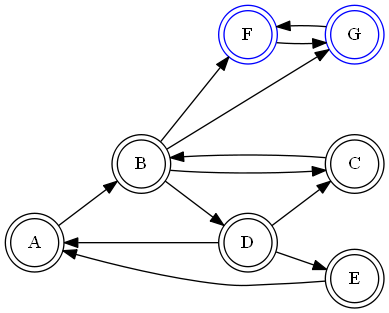


Figure 5: A network with two “sink” nodes, F and G.

Google’s inventors decided to modify the PageRank algorithm to reflect that a user occasionally clicks on a random link or a link from their list of bookmarks, and that users rarely spend a large amount of time on webpages without useful outlinks. We add a “damping factor” α that represents the probability a user will click on a random link, and will *not* follow the weighted PageRanks we computed in the previous table. The formula represents a model of a web surfer who gets bored after several clicks and switches to a random page. The PageRank value of a page reflects the chance that the random surfer will land on that page by clicking on a link. If a page has no links to other pages, it becomes a sink and therefore terminates the random surfing process. If the web surfer arrives at a sink page, it picks another page *at random* and continues surfing again. Our previous equation becomes slightly more complicated, and is given here:

where is the damping factor and is the total number of pages in the network. From experience, it is known that a damping factor of 0.85 closely approximates reality, so we will use that in our computation.

# Deliverables and Due Date

This assignment is in part to teach you an artificial intelligence technique, and in part to help me make sure you have the correct version of Python installed, and know the basics of how to program in Python. Therefore, I would like you to follow the skeleton code given here: XXX, and not try to solve this assignment with advanced Python packages that may be available to do the work for you. You are welcome to work together on this assignment, and can share code with each other – as long as you understand what you are writing!

**DUE DATE: at the beginning of the first day of class**

**DELIVERABLES:  
1) Python code used to compute your network’s PageRank values for each node  
2) Screenshots or similar documentation showing Python package versions and your PageRank numbers**