Inf 674: Importance Diffusion and PageRanking

Nidhi Hegde Fabien Mathieu

February 11, 2016

You already learned the theoretical basis of PageRank with M. Lelarge. In this session, you will get your hands dirty.

1 Sandbox

To start with, a small experiment that plays with the assumptions of Perron-Frobenius theorem.

Consider the following game: you have a circular game board made of $n = 36$ squares numbered from 0 to 35. At (discrete) turn $t = 0$, a player stands on square 0. Every turn, the player tosses an unbiased coin. If it is a head, she moves forward $a = 1$ step. If it is a tail, she moves forward $b = 2$ steps.

Question 1

Write a small program that displays an animation the evolution of the distribution of the player on the squares as t increases. You can quicklearn animations by looking at the examples from `http://matplotlib.org` (I used `simple_anim`).

Comment what you observe.

Question 2

We assume now $a = 1$, $b = i^2$, where i is the number of the square where the player stands. Observe and comment.

Question 3

Same with $a = 1$, $b = 7$.

Question 4

Same with $a = 2$, $b = 4$.

Question 5

With probability 1/10, the coin lands on its edge (it is a very thick coin). Default procedure: toss again until is does not land on its edge. Exception: if it lands on its edge while the player stands on square 35, she realizes that the game is pointless and quits the game. Redo previous questions with this new rule.

Question 6

Test (carefully) the values of n that you can reasonably use with your current program and computer.

2 Playground

Let us continue by testing some variants of PageRank on some artificial graphs, so we can understand the interest of damping the random walk.

Consider the graph G_h built as follows (you can use $h = 3$ to test what you do, $h = 9$ for practical evaluation): take $n = 2^{h+1} - 1$ nodes numbered from 0 to $2^{h+1} - 2$ and add the following edges:

- For all $i \geq 1$, add one edge from i to $\lfloor (i-1)/2 \rfloor$ and one from i to $i-1$ (if it does not exist yet).
- Node 0 has one edge to each node j such that $j \in [2^h - 1, 2^{h+1} - 2]$.

Question 1

Try to guess the most important nodes of the graph according to the PageRank random walk. Explain your guess → just copying answers from the next question is not enough.

Question 2

Compute the PageRank defined as solution of $P = PA$, where A is the stochastic matrix associated to G_h ($A(i,j) = 1/deg_+(i)$ if $i \to j$, 0 otherwise). You will proceed iteratively, starting from $P = \mathbf{1}/n$ and iterating $P \leftarrow PA$ until $||P - PA||_1 < \epsilon$ (recommendation: $\epsilon = 10^{-8}$). You'll display

- for the first 10 iterations the top ten nodes according to the current P;
- the total number of iterations;
- the final top ten nodes.

As a fail-safe, fix a maximal number of iterations (recommendation: 2000). If reached, signal you failed to converge and give the top ten nodes from the last value of P.

Question 3

We add to G_h b additional nodes ($b = 10$) numbered from n to $n + b - 1$. Each new node i has an incoming edge from $i - 1$.

- Start over the previous questions on the new graph. What happens?
- Use $P \leftarrow PA/||PA||_1$ for the update. What happens?

Question 4

We add one single edge to the previous graph, from $n + b - 1$ to $n + b - 2$.

- Start over the previous questions on the new graph. What happens?
- Use $P \leftarrow dPA + (1-d)\mathbf{1}$ for the update. What happens?
- Discuss the number of iterations and the rankings obtained on the different graphs and PageRank variants.

3 Battlefield

Now the real deal: you will learn how to use PageRank on a crawl of French Wikipedia made in 2013 and available on `http://webgraph.di.unimi.it/`. The graph has been "cleaned": only links from one article to another article are kept.

For this part, you will need three files:

- `frwiki-2013.ids` contains the article titles (one per line, n lines in total). By convention, each article is associated to its line number (from 0 to $n-1$).
- `frwiki-2013.adja` contains the adjacency list of the graph: line i (from 0 to $n-1$) contains, in plain ASCII, the numbers of the articles that are linked by i.
- `frwiki-2013-t.adja` contains the adjacency list of the transposed graph: line i (from 0 to $n-1$) contains the numbers of the articles that have a link to i.

Files are available on a USB key. They can also be downloaded at `h t t p s : / / g o f i l e . m e /2esZs/EdkejZfw`.

The python corner:

- Process a file line per line, in order:

```
with open(nomfichier) as f: # File access
        for line in f: # Loop over the lines
                (do stuff there)
```

- The title file is utf-8. Try `import codecs` and `codecs.open("frwiki-2013.ids", "r", "utf-8")`.
- Turn a sequence of ASCII numbers from string `line` into an array:

```
[int(s) for s in line.split()]
```

- Test if a string `str1` is included in a string `str2` : `str1 in str2`

Question 1

First try and tame the files. Compute the number of nodes n, the number of edges m, maximal indegree and maximal outdegree.

Question 2

Propose a function that takes for input a vector P of size n and an integer $k > 0$ (default to $k = 20$) and returns the title of k articles with highest value according to P. Test your function by displaying the names of the k articles with highest indegree.

Question 3

Compute a PageRank based on the following pseudo-code:

```
PARAMETERS: File prefix = "frwiki-2013", d = .7, precision = .1

INITIALISATION :
D <- Outdegree vector
Adjt <- Transposed adjacency list
Pold <- [1,...,1]
P <- [1,...,1]
Z <- [1,...,1]
Error <- 1.0
```

```
ITERATIONS
while Error > precision:
        Pold <- P
        for j in range n:
                P[j] <- d * SUM(Pold[k]/deg[k] for k->j) + Z[j]
        Error <- ||P-Pold|| / ||P||

Return P
```

You will monitor the time required for the successive initialization steps (mostly degree and adjacency), the time per iteration and the total time (`import time`, `time.clock()`).

Question 4

The algorithm above may be a little bit slow. Let's do some tuning. The optimization proposed below are not mandatory for the next question, but they are recommended. You will learn to speed up your algorithm by mixing theoretical and practical considerations, and it will allow you to play more with the next question. Don't forget to compare your results to the previous question to verify that you did not break anything.

1. Playing with d and with the precision may help to speed up the process. Is it a good idea?
2. Look how much RAM you use. If it is more than 1 Go, you probably are too brutal in the way you load the adjacency matrix. You should be able to store your adjacency list in one integer np.array of size $n + m$ with no impact on speed.
3. Downsize the initialization delay. You do not want to parse huge text files anytime you compute a PageRank. Use `np.save('myfile', x)` and `x = np.load('myfile.npy')` for degree and adjacency. Even better: test on the fly if a `.npy` file exists:

```
try:
        x = np.load(filethatcontainsx+".npy")
except IOError:
        (make variable x)
        np.save(filethatcontainsx, x)
```

4. Downsize the time per iteration. If you coded the algorithm above exactly, you probably perform m divisions per iteration. You should be able to cut that to n divisions per operations (with better memory access too). Beware of leaves! How does the time per iteration evolve?
5. Downsize the time per iteration. Do the update in place instead of using *Pold*. How does the number of iteration evolve (you probably need to use a better precision to observe the difference). For the record, this is called the Gauss-Seidel method, and one can prove that it enhances convergence.
6. For practical use, the ranking matters but not the actual importance values. Replace the stopping condition by *last iteration did not change the ranking of the top k articles* ($k = 20$). If you did the previous optimization, you probably do not need *Pold* anymore...

7. Instead of using $[1, \ldots, 1]$ as initial value, why not use a previously saved computation? Interest: if you already computed a PageRank for a small k and want better precision (larger k), you do not need to start over. For the record, this is how one updates PageRank of Web graphs when a fresh crawl is made: one uses the old PageRank as an educated guess for the new PageRank.
8. Can you avoid the use of the transposed adjacency matrix (for experts only)?

Question 5

The algorithm uses a uniform vector $Z = [1, \ldots, 1]$. Modify it to take an additional argument `s` so $Z[i]$ is 1 if `s` is in the title of i, 0 otherwise. Save your Z and P in files with `s` in the name. Remark: using a non-uniform Z will create a bias in favor of the pages that contain `s` in the title. A simple workaround consists in ranking according to $P - Z$ instead of P ($P - Z$ is called the residual PageRank). Test your algorithm on some values of `s` and comment. Possible inputs: `film`, `biologie`, `physique`, `philo`, ... Comment the results!

Question 6

If you made it so far and your optimizations are descent enough, you can try the English Wikipedia (three times bigger).