The PageRank algorithm

Application of *Probability spaces, events, conditioning*

PageRank is the algorithm at the core of Google's system for measuring the relevance of search results. Although Google now uses a combination of different ranking systems, the original algorithm is still one of them, and is implemented by other companies as well. See PageRank's Wikipedia page for more context.

To understand the basic idea of this algorithm, one only needs what we have discussed in Part 1 of the course. If you are just a bit curious, you can read Section 1 below and stop there. If you want to see an example, you can read Section 5; you will only need the very short definitions of E, D, P and μ from the beginning of Section 4. The last few paragraphs require some knowledge of linear algebra, that you can safely ignore if you feel like it. If you are interested in how one applies the concepts of the course to this concrete situation, you can read Sections 2 to 4, and maybe try to apply it yourself to Section 5, see if you get the same results. For an arbitrary collection of pages, this procedure belongs to a class of processes called *Markov chains;* the general theory of Markov chains is well known to the experts, and a review of some of its first results is given in Section 6.

1 The random internet user

The rough idea behind PageRank is that a collection of pages will be ranked according to how much time a user clicking randomly on links will spend on each page. Intuitively, if a page P is the target of many links, it means that many pages think that P is relevant. Since the surfer has a higher chance of getting to P the more pages link to it, it may give a good idea of the relevance of P.

The precise model is as follows. Suppose you have a collection of pages, say P_1 to P_k. We consider it as the collection of vertices of a graph. If there is at least one link from a page to another, we draw a (directed) arrow from the source to the target; we ignore self links. If a page has no arrows coming from it, in other words it links to no other page, it is called a *sink*. Sinks are problematic, because our surfer will be stuck on this page. To resolve this issue, we add for every sink all the possible arrows coming from it, as if it was linking every page in the collection. We also add an arrow from the vertex to itself, which is the only case where we can have arrows from a page to itself. This gives us a graph as in Figure 2b, Section 5, for instance. Choose a number $0 < d < 1$ (originally, Google used $d = 0.85$); we call it the *damping parameter*.

The user starts on a given page. It turns out this choice is not important: choosing the first one, or the one with the most links, will not make a difference. In fact, Google used to choose the initial page uniformly at random. At each step, the user will jump from one page to another, by applying the following procedure. All the choices we make are independent.

- With probability d, we choose an arrow leaving the current page uniformly at random, and follow this arrow. The idea is that the user clicks a random link. Note that there is at least one arrow, according to the way we dealt with sinks.
- With probability $1 - d$, the user gets bored of clicking, and just selects a page uniformly at random, with no consideration for the arrows.

The second item is important here, because if the collection of pages can be separated into two groups with no link between them, then the user will get stuck in one half of the graph, potentially ignoring pertinent results.

After many steps (say $N = 20$), we can consider the probabilities p_i that the user is on page P_i. This probability measures how well the page scores on our scale: the higher the probability, the more relevant we expect the page to be.

One can show, see the section about Markov chains, that the probabilities p_i converge when N goes to infinity; that's why it makes sense to consider N very large. Moreover, it turns out that this limit is equal to the proportion of time the user spends on a given page. This gives another interpretation of the final ranking: a page scores higher the more time the user spends on it.

2 A variant on the law of total probability

In the following, we will need the following variation on the law of total probability. Suppose $A_1, \ldots, A_k$ is a partition of S, i.e. every outcome $s \in S$ belongs to precisely one A_i. Let A and B be any other events. Then, provided the events we condition upon do not have probability zero, we have

$$\mathbb{P}(B|A) = \mathbb{P}(B|A \cap A_1)\mathbb{P}(A_1|A) + \mathbb{P}(B|A \cap A_2)\mathbb{P}(A_2|A) + \cdots + \mathbb{P}(B|A \cap A_k)\mathbb{P}(A_k|A).$$

The one we saw in class corresponds to $A = S$.

Here is a proof of this version with some details missing, that I am confident you can fill in yourself. Notice that

$$\mathbb{P}(B|A \cap A_i)\mathbb{P}(A_i|A) = \mathbb{P}(B \cap A_i|A).$$

Since

$$B = (B \cap A_1) \cup \cdots \cup (B \cap A_k)$$

where the union is disjoint, the axioms of probability applied to $\mathbb{P}(-|A)$ yield

$$\mathbb{P}(B|A) = \mathbb{P}(B \cap A_1|A) + \mathbb{P}(B \cap A_2|A) + \cdots + \mathbb{P}(B \cap A_k|A),$$

which concludes.

3 Probabilities of jumping

We write $E_n(i)$ for the event

"The user is on page i after the nth step."

For instance, if the user is initially on page 1, then $\mathbb{P}(E_0(1)) = 1$ and $\mathbb{P}(E_0(i)) = 0$ for any other page P_i. If on the other hand we chose the starting point uniformly at random, then $\mathbb{P}(E_0(i)) = 1/k$

for each of the k pages we consider. We want to know $\mathbb{P}(E_N(i))$ for N large; it will give us our ranking for page P_i.

Suppose we know all the $\mathbb{P}(E_n(i))$ for some fixed n; for instance, we just described the case $n = 0$. Then we can use the (usual) law of total probability to get

$$\mathbb{P}(E_{n+1}(j)) = \mathbb{P}(E_{n+1}(j)|E_n(1))\mathbb{P}(E_n(1)) + \cdots + \mathbb{P}(E_{n+1}(j)|E_n(k))\mathbb{P}(E_n(k)).$$

Provided we know the probabilities $\mathbb{P}(E_{n+1}(j)|E_n(i))$, then hopefully we can compute the $\mathbb{P}(E_n(i))$ iteratively. Note that we do not have to do it all by hand; Google has access to powerful computers.

Let P_i and P_j be two pages (possibly the same, i.e. $i = j$). We want to compute the probability $\mathbb{P}(E_{n+1}(j)|E_n(i))$. In other words, we are looking for the probability to get from the first to the second when we perform step $n + 1$.

We consider two ways to get from one page to the other.

- If there is an arrow from P_i to P_j, then provided the surfer decides to click a link (with probability d), there is a probability $1/D_i$ to get from i to j, where D_i is the number of arrows from i to any other vertex. (D_i is called the *outdegree* of i.)
- If the user decides to jump randomly rather than clicking (with probability $1 - d$), there is a probability $1/k$ that j is the page chosen uniformly at random by the user.

Mathematically, we use the law of total probability and get

$$\begin{aligned}\mathbb{P}(E_{n+1}(j)|E_n(i)) &= \mathbb{P}(E_{n+1}(j)|E_n(i) \cap \text{“bored at step } n+1\text{”})\mathbb{P}(\text{“bored at step } n+1\text{”}|E_n(i)) \\ &\quad + \mathbb{P}(E_{n+1}(j)|E_n(i) \cap \text{“not bored at step } n+1\text{”})\mathbb{P}(\text{“not bored at step } n+1\text{”}|E_n(i)) \\ &= \frac{1}{k} \cdot (1-d) + \mathbb{P}(E_{n+1}(j)|E_n(i) \cap \text{“not bored at step } n+1\text{”}) \cdot d,\end{aligned}$$

where, depending on the status of P_i and P_j, we have

$$\mathbb{P}(E_n(j)|E_{n-1}(i) \cap \text{“not bored at step } n\text{”}) = \begin{cases} 1/D_i & \text{if there is an arrow from } P_i \text{ to } P_j, \\ 0 & \text{else.} \end{cases}$$

Note that $\mathbb{P}(E_{n+1}(j)|E_n(i))$ does not depend on n.

4 Transition matrix

One very efficient way to store all this information about the probabilities is to store it in a matrix. We call P the matrix whose (i, j) entry is the probability $\mathbb{P}(E_n(j)|E_{n-1}(i))$. Then, according to the above,

$$P = \frac{1-d}{k}E + dD,$$

where E is the matrix of size $k \times k$ full of ones, and the (i, j) entry of D is $1/D_i$ if there is an arrow from i to j, 0 otherwise. The matrix P is called the *Google matrix*; it is a particular example of a *transition matrix*, see the section about Markov chains.

Let us get back to trying to compute $\mathbb{P}(E_n(i))$. Define

$$\mu(n) = \begin{pmatrix} \mathbb{P}(E_n(1)) & \cdots & \mathbb{P}(E_n(k)) \end{pmatrix}$$

the (row) vector of these probabilities. As discussed before, hopefully we can compute $\mu(n)$ iteratively using the law of total probability:

$$\begin{aligned}\mu(n+1)_j &= \mathbb{P}(E_{n+1}(j)) \\ &= \mathbb{P}(E_{n+1}(j)|E_n(1))\mathbb{P}(E_n(1)) + \cdots + \mathbb{P}(E_{n+1}(j)|E_n(k))\mathbb{P}(E_n(k)) \\ &= P_{1j}\mu(n)_1 + \cdots + P_{kj}\mu(n)_k \\ &= (\mu(n)P)_j,\end{aligned}$$

where in the last line $\mu(n)P$ denotes the matrix product. This means that $\mu(1) = \mu(0)P$,

$$\mu(2) = \mu(1)P = (\mu(0)P)P = \mu(0)P^2,$$

and iteratively $\mu(n) = \mu(0)P^n$.

Although matrix products are tedious to compute by hand, it is such a common operation in algorithm design that processors are especially build to be efficient at such computations. This is a good thing, because ranking Google results means running this matrix multiplication over millions of pages.

5 An example on 5 pages

The following graph represents a small collection of pages. Every arrow represents a link from a page to another. Note that we can have several links to the same page (e.g. P_2 to P_1), or even self links (e.g. P_4).

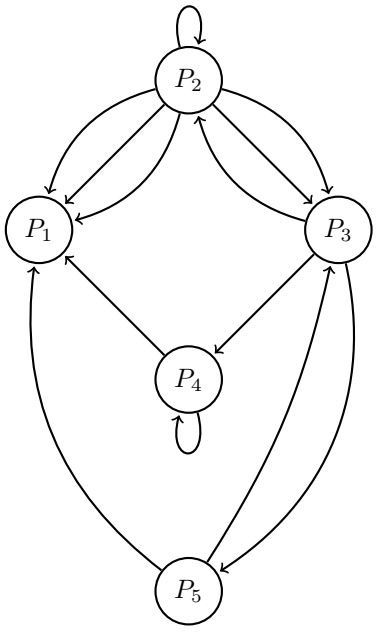

Figure 1. — A baby internet

To get the graph described in Section 1, starting from a graph without edges, first we place an arrow from P_i to P_j ($i \neq j$) if there was such an arrow in the initial graph (left picture), then we add arrows connecting the sinks (in this case only P_1) to every vertex, including itself (right). The D_i are the outdegrees: the number of arrows going out a given vertex.

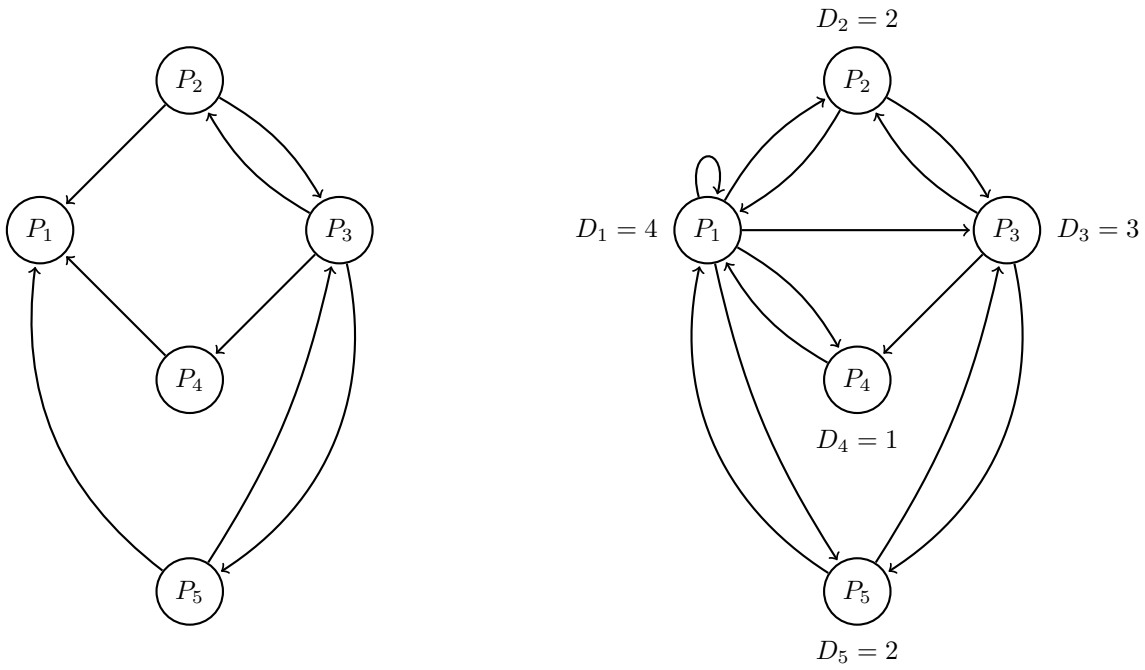

Figure 2. — Induced graphs

We could compute the probabilities $\mathbb{P}(E_{n+1}(j)|E_n(i))$ by hand; for instance, using the law of total probability,

$$\begin{aligned}\mathbb{P}(E_{n+1}(2)|E_n(3)) &= \mathbb{P}(E_{n+1}(2)|E_n(3) \cap \text{“not bored at step } n+1\text{”})\mathbb{P}(\text{“not bored at step } n+1\text{”}|E_n(3)) \\ &\quad + \mathbb{P}(E_{n+1}(2)|E_n(3) \cap \text{“bored at step } n+1\text{”})\mathbb{P}(\text{“bored at step } n+1\text{”}|E_n(3)) \\ &= \frac{1}{3} \cdot d + \frac{1}{5} \cdot (1-d).\end{aligned}$$

Using the reasoning above, though, we get directly

$$E = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \qquad D = \begin{pmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 1/3 & 0 & 1/3 & 1/3 \\ 1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \end{pmatrix},$$

from which we deduce

$$P = \frac{1-d}{5}E + dD = \begin{pmatrix} \frac{1-d}{5}+\frac{d}{5} & \frac{1-d}{5}+\frac{d}{5} & \frac{1-d}{5}+\frac{d}{5} & \frac{1-d}{5}+\frac{d}{5} & \frac{1-d}{5}+\frac{d}{5} \\ \frac{1-d}{5}+\frac{d}{2} & \frac{1-d}{5} & \frac{1-d}{5}+\frac{d}{2} & \frac{1-d}{5} & \frac{1-d}{5} \\ \frac{1-d}{5} & \frac{1-d}{5}+\frac{d}{3} & \frac{1-d}{5} & \frac{1-d}{5}+\frac{d}{3} & \frac{1-d}{5}+\frac{d}{3} \\ \frac{1-d}{5}+1 & \frac{1-d}{5} & \frac{1-d}{5} & \frac{1-d}{5} & \frac{1-d}{5} \\ \frac{1-d}{5}+\frac{d}{2} & \frac{1-d}{5} & \frac{1-d}{5}+\frac{d}{2} & \frac{1-d}{5} & \frac{1-d}{5} \end{pmatrix}.$$

Note that the $(3,2)$ entry is indeed what we found. For $d = 0.85$, we get

$$P \simeq \begin{pmatrix} 0.2000 & 0.2000 & 0.2000 & 0.2000 & 0.2000 \\ 0.4550 & 0.0300 & 0.4550 & 0.0300 & 0.0300 \\ 0.0300 & 0.3133 & 0.0300 & 0.3133 & 0.3133 \\ 0.8800 & 0.0300 & 0.0300 & 0.0300 & 0.0300 \\ 0.4550 & 0.0300 & 0.4550 & 0.0300 & 0.0300 \end{pmatrix}.$$

Using a computer, we can compute a few values for $\mu(n)$ (as a column for space saving):

$$\mu(0)^T = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}, \quad \mu(1)^T = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}, \quad \mu(2)^T = \begin{pmatrix} 0.40400 \\ 0.12066 \\ 0.23400 \\ 0.12066 \\ 0.12066 \end{pmatrix}, \quad \mu(5)^T = \begin{pmatrix} 0.32729 \\ 0.15442 \\ 0.20942 \\ 0.15442 \\ 0.15442 \end{pmatrix},$$

$$\mu(10)^T = \begin{pmatrix} 0.34135 \\ 0.14806 \\ 0.21446 \\ 0.14806 \\ 0.14806 \end{pmatrix}, \quad \mu(15)^T = \begin{pmatrix} 0.34026 \\ 0.14855 \\ 0.21407 \\ 0.14855 \\ 0.14855 \end{pmatrix}, \quad \mu(20)^T = \begin{pmatrix} 0.34034 \\ 0.14851 \\ 0.21410 \\ 0.14851 \\ 0.14851 \end{pmatrix}, \quad \mu(21)^T = \begin{pmatrix} 0.34033 \\ 0.14852 \\ 0.21409 \\ 0.14852 \\ 0.14852 \end{pmatrix}.$$

For each iteration, we multiply the vector by P on the right. As one can see, it becomes fairly stable for $N = 20$. Moreover, if we change the initial point, we get more or less the same result: for instance,

$$\mu(0)^T = \begin{pmatrix} 0.0 \\ 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}, \quad \mu(1)^T = \begin{pmatrix} 0.455 \\ 0.030 \\ 0.455 \\ 0.030 \\ 0.030 \end{pmatrix}, \quad \mu(2)^T = \begin{pmatrix} 0.15835 \\ 0.23626 \\ 0.13285 \\ 0.23626 \\ 0.23626 \end{pmatrix}, \quad \mu(5)^T = \begin{pmatrix} 0.38335 \\ 0.12904 \\ 0.22951 \\ 0.12904 \\ 0.12904 \end{pmatrix},$$

$$\mu(10)^T = \begin{pmatrix} 0.33700 \\ 0.15003 \\ 0.21290 \\ 0.15003 \\ 0.15003 \end{pmatrix}, \quad \mu(15)^T = \begin{pmatrix} 0.34060 \\ 0.14840 \\ 0.21419 \\ 0.14840 \\ 0.14840 \end{pmatrix}, \quad \mu(20)^T = \begin{pmatrix} 0.34032 \\ 0.14852 \\ 0.21409 \\ 0.14852 \\ 0.14852 \end{pmatrix}, \quad \mu(21)^T = \begin{pmatrix} 0.34035 \\ 0.14851 \\ 0.21410 \\ 0.14851 \\ 0.14851 \end{pmatrix}.$$

Using this procedure, we decide that the more important pages are, in order, P_1, P_3, and then every other pages with the same value. It makes sense that P_1 should be ranked fairly high, since

a lot of pages are linking to it; it means many pages think it is relevant. Usually, pages will not have the same rank; it is a coincidence due to the fact that we have very few pages.

If you are familiar with linear algebra, you can use whatever scientific software you like to compute the eigenvalues of P. You'll see that one of them is precisely one, whereas all the others (say $c_2, \dots, c_5$) have magnitude less than one. It means that P admits a unique vector λ that is (right) invariant, i.e. $\lambda P = \lambda$ (consider the eigenvectors of the transpose). Decomposing μ as a linear combination of λ and the other eigenvectors[1] (say $v_2, \dots, v_5$), we have $\mu(0) = m\lambda + a_2 v_2 + \cdots + a_5 v_5$, and

$$\mu(n) = \mu(0)P^n = m\lambda P^n + a_2 v_2 P^n + \cdots + a_5 v_5 P^n = m\lambda + c_2^n v_2 + \cdots + c_5^n v_5.$$

Since the $c_2, \dots, c_5$ have moduli less than one, the last 4 terms contribute an exponentially small amount, whereas $m\lambda$ stays constant. This means that $m\lambda$ is in fact very close to our $\mu(20)$ computed above. From any initial condition, we will in fact converge to the same $m\lambda$. It is a bit more subtle to see that m is always the same, and that $m\lambda$ is a probability function (only real non-negative entries, summing to one), from linear algebra alone. It is more or less the content of the Perron–Frobenius theorem, and it would drive us largely beyond the scope of this document.

6 About Markov chains

In their simplest form, Markov chains are processes over a finite collection of states (for example results from a Google search). At a given time n, we are at a (random) state i, and in the next step, we jump from i to j with a certain probability, given by a transition matrix as in Section 4. We can define, as in Section 3, the events $E_n(i)$ to be in state i at time n, and given initial probabilities $\mu(0) = (\mathbb{P}(E_0(1)), \dots, \mathbb{P}(E_0(k)))$, we can deduce iteratively the probabilities $\mu(n) = (\mathbb{P}(E_n(1)), \dots, \mathbb{P}(E_n(k)))$ to be in a given state at time n. It is given by the same formula as in Section 4, $\mu(n) = \mu(0)P^n$, where P is the transition matrix.

An example commonly found in probability books is the following. Consider a knight alone on a chess board. The knight starts from a a given square, then at each step jumps to one of the squares it can access, uniformly and independently of its previous choices. After some time, is it possible to predict the position of the knight? Which squares did it visit most? How much time will we wait before it returns to its initial square? These questions are typical when one studies Markov chains, and the answers are well known to the experts.

Let us state the results we discussed above in the case of PageRank. We say that the transition matrix P is 1-irreducible if its entries are all positive, including on the diagonal; this is always the case for PageRank, since the probability to get from i to j is at least the probability that the user gets bored and jumps to j, which has probability $(1-d)/k > 0$.

Theorem 1. *Consider a Markov chain with transition matrix P, and suppose P is 1-irreducible.*[2]
Then there exists a unique vector λ such that $\lambda = \lambda P$, and for all state i, the probability $\mathbb{P}(E_n(i))$ of being in state i at times n converges to λ_i. Moreover, the vector λ is a probability function.

In the algorithm, we compute $\mu(n)$ for n large, but in fact it is λ we are looking for: the value λ_i is the score of page P_i, but computing this exact value is unfeasible for large collection of pages, so we settle for the approximation $\mu(n)$.

[1] Let us glance over the fact that P is not diagonalisable; it truly doesn't matter.

[2] We need a lot less than this; the key words here are *aperiodicity* and *irreducibility*, for existence and convergence respectively.

I claimed in Section 1 that λ_i is also the average time spent by the used on the page P_i. This is a consequence of the so-called ergodic theorem for Markov chains.

Theorem 2. *Consider a Markov chain with transition matrix P, and suppose P is 1-irreducible.*[3]

Choose an initial state (possibly random), and consider $N_n(i)$ the number of times the chain is in state i up to step n. Then $N_n(i)/n$ converges to λ_i, for λ the unique vector such that $\mu = \mu P$.

[3] Again, aperiodicity and irreducibility would do.