Lecture 17: Markov Chains, continued:Analysis of regular Markov Chains

CSCI2244-Randomness and Computation

May 1, 2019

1 Regular Markov Chains

A Markov chain (S, M) is *regular* if some power M^k has strictly positive entries (*i.e.,* no zero entries).

Note that our lily pad example is a regular Markov chain. While M itself contains zero entries, as we saw in the last notes, M^2 has no zero entries.

The main property of regular Markov chains is the existence of a limit for the powers M^r, just as we saw with the lily pad example. We will not prove this theorem, but instead just give the statement:

Theorem 1 *Let M be the $n \times n$ transition matrix of a regular Markov chain.*

$$\lim_{k \to \infty} M^k$$

exists.

Denote this limit matrix by M_∞. All the rows of M_∞ are equal. Further, if $\mathbf{w}$ is the $1 \times n$ matrix (row vector) representing this row, then

$$\mathbf{w}M = \mathbf{w},$$

and $\mathbf{w}$ is the unique solution to this equation with the property that its entries add to 1.

The row vector $\mathbf{w}$ is called the *stationary distribution* of M. What this means is that in the long run, the probability of being in any state j after a large number of steps is independent of the starting state.

I don't want to leave you with the impression that every Markov chain is either absorbing or regular. Consider, for example, the chain given by this matrix:

$$M = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

We have $M^k = M$ if k is odd, and $M^k = I$ if k is even, so $\lim_{k\to\infty} M^k$ does not exist. This represents the behavior of a frog on a pond with two lily pads, who just hops back and forth between the two at every time step. Although there is no limiting distribution, the vector

$$\mathbf{w} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

is fixed by M.

2 Computing the stationary distribution

2.1 Manually

We can compute the stationary distribution by solving the system of linear equations

$$\mathbf{w} \cdot M = \mathbf{w},$$

or, equivalently

$$\mathbf{w}(I - M) = \mathbf{0},$$

where $\mathbf{0}$ denotes the $1 \times n$ matrix all of whose entries are 0.

This system of equations always has $\mathbf{0}$ itself as a solution, but when M is the transition matrix of a regular Markov chain, it has nonzero solutions, all of which are constant multiples of each other. We have to divide each component of the solution by the sum of the components to get the unique solution for which the component sum is 1.

Let's illustrate this with the lily pad example. We have

$$I - M = \begin{bmatrix} 1 & -1/2 & -1/2 \\ -1/2 & 1 & -1/2 \\ -1/3 & -1/6 & 1/2 \end{bmatrix}.$$

So we have to solve

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 1 & -1/2 & -1/2 \\ -1/2 & 1 & -1/2 \\ -1/3 & -1/6 & 1/2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix},$$

or, what is the same thing,

$$\begin{array}{rcrcrcl} x & - & y/2 & - & z/3 & = & 0 \\ -x/2 & + & y & - & z/6 & = & 0 \\ -x/2 & - & y/2 & + & z/2 & = & 0. \end{array}$$

As mentioned, any constant multiple of a solution is also a solution, and there must be a solution in which all the entries are strictly positive. So there must be a solution with $x = 1$. If we substitute $x = 1$ in the first two equations above, we get a smaller system that is easy to solve by hand. You can verify that the solution is

$$x = 1, y = 0.8, z = 1.8.$$

This is *not* the stationary distribution, because the components do not add to 1. We obtain the stationary distribution by dividing each of these components by their sum $1 + 0.8 + 1.8 = 3.6$ to get

$$x = 5/18 = 0.27778, y = 2/9 = 0.22222, z = 0.5,$$

which is what we observed before.

2.2 With `numpy`

There are several options for automatic computation of the stationary distribution. Solving the system $\mathbf{w}(I - M) = \mathbf{0}$ directly will not work: Due to roundoff error,

the system will treat this as a system with a unique solution and probably return **0** as the answer.

One simple solution is an iterative method, where we repeatedly compute larger and larger powers M^k of the transition matrix until the all the values in each column are within a desired small distance of one another. This can be speeded up by repeatedly squaring the matrix, that is, by computing

$$M, M^2, (M^2)^2 = M^4, (M^4)^2 = M^8,$$

etc., so that we do only k matrix multiplications to compute M^{2^k}.

A fancier method is to compute the so-called eigenvalues and eigenvectors of the transposed matrix of M. You don't have to know what these terms mean! A call in `numpy` to

```
eig(transpose(m))
```

where m is an $n \times n$ matrix, returns a tuple with two components: The first is a length-n array of numbers (these are the eigenvalues), and the second is an $n \times n$ matrix. One of these numbers is guaranteed to be 1 if you started with a regular Markov chain. The second component is an $n \times n$ matrix. The column of this matrix corresponding to the eigenvalue 1 is a solution to our equation $\mathbf{w} \cdot M = \mathbf{w}$. Again, this is not, in general, the stationary distribution. But if we divide this by the sum of the components of **w**, we obtain the desired solution.

We illustrate this with the lily pad example: the computation is shown below. Note that the eigenvalue 1 occurs in component 0 of `u[0]`, so that when we create the variable `v`, we assign to it column 0 of the matrix `u[1]`.

```
>>> m=array([[0,0.5,0.5],[0.5,0,0.5],[1./3,1./6,0.5]])
>>> u=eig(transpose(m))
>>> u[0]
array([  1.00000000e+00,  -5.00000000e-01,  -7.48988743e-18])
>>> u[1]
array([[ -4.52678730e-01,  -7.07106781e-01,  -2.67261242e-01],
       [ -3.62142984e-01,   7.07106781e-01,  -5.34522484e-01],
       [ -8.14821714e-01,  -6.23219423e-17,   8.01783726e-01]])
>>> v=[u[1][j][0] for j in range(3)]
>>> v
[-0.45267873021259269, -0.36214298417007379, -0.81482171438266693]
>>> v/sum(v)
array([ 0.27777778,  0.22222222,  0.5       ])
```

It is possible that when you do this, the numbers returned by `eig` will be *complex numbers,* which have an additional *imaginary component,* and look like

```
2.432917 + 1.43e-26j
```

(In these examples, the imaginary component will always be very small; the fact that it is there at all is due to roundoff error.) When this occurs, you need to replace `u[1]` by `real(u[1])`, which will get rid of the imaginary part, or simply ignore the imaginary part.

3 The Page Rank Algorithm

This method for computing an 'importance' ranking for Web pages was at the heart of the original Google search engine, created by Larry Page and Sergei Brin in 1998. (The name of the algorithm is a pun, referring to both the creator of the algorithm and what it does.) Google, it would seem, has moved beyond the basic Page Rank algorithm to rank the results returned by a Web search, but the algorithm can be useful in other sorts of citation networks (for example, measuring the impact of a scientific paper). And it provides a good illustration of a practical use of regular Markov chains.

One of the explanations Page and Brin gave for the method is this: Imagine that the Web (the whose Web!) consists of N pages $1, \ldots, N$. Imagine a 'random surfer' who starts out on a page, selects an outgoing link at random, clicks on the link, and then repeats this with the page he lands on, etc. Where does the surfer spend the largest amount of his time?

If we let m_i denote the number of outgoing links on page i, then the surfer on page i jumps to page j with probability

$$a_{ij} = \begin{cases} 1/m_i, & \text{if there is a link from } i \text{ to j,} \\ 0, & \text{otherwise.} \end{cases}$$

This looks like the transition matrix of a Markov chain. There are a few problems, however. If a page contains no outgoing links, then the matrix entry a_{ij} is undefined. Even if we were to remove such pages, the resulting chain is not in general regular. For instance, we could have two pages that just link to one another, like our example above of the frog on a pond with two lily pads; if the surfer starts at one, he will just jump back and forth between the two.

So Brin and Page added a method to get out of such sinks: This is a 'damping factor' $d < 1$. With probability d, the surfer's behavior is governed by the rules

above. But with probability $1 - d$, the surfer gets bored and jumps to a random page one the Web. So the matrix is given by

$$a_{ij} = \begin{cases} d/m_i + (1-d)/N, & \text{if there is a link from } i \text{ to j,} \\ (1-d)/N, & \text{otherwise.} \end{cases}$$

If page i has no outgoing links, then we can set $a_{ij} = 1/N$ for all j. (There is some variation in the description of how to handle this case.)

The $N \times N$ matrix M whose (i, j)-entry is a_{ij} is now a regular Markov chain. Thus it has a stationary distribution, which gives the relative amounts of time that the surfer spends on each page of the Web.

Two practical considerations: First, even in 1998, the Web consisted of more than 100 million = 10^8 pages. Thus the full matrix would have 10^{16} entries, and multiplying two such matrices would require 10^{48} individual operations, which is effectively impossible, even for Google. So to compute the stationary distribution, Google had to rely on iterative methods. Still, this was a gargantuan task, requiring a complete crawl of the Web to be repeated at regular intervals to update the ranking.

Second, how should the value d be chosen? There are several considerations: You want the method to work well, in the sense that what the algorithm ranks as important really does correspond to an intuitive notion of importance of a web page, and you would also like to get relatively rapid convergence to the limit, so that the iterative method of computing the stationary distribution is reasonably efficient. Brin and Page settled on $d = 0.85$ as a good value to use.