Markov Chains

Akshath Jain & Deepayan Patra

December 2018

1 Overview & History of PageRank

Invented by Andrei Markov and published in 1906, Markov Chains are collections of random variables that have the explicit property that given the present, the future is conditionally independent of the past.[1] With this mathematical construct, Google co-founders Larry Page and Sergey Brin proposed that Markov Chains could be used to efficiently rank the popularity, relevance, and legitimacy of web pages. Deployed in 1998, this PageRank algorithm was far superior and more efficient than other approaches, which propelled Google to the forefront of search.[2] This paper analyzes the fundamental aspects of the PageRank algorithm.

Beyond PageRank, Markov Chains have many applications across all fields, including biology, finance, and game theory. Specifically, this can be used for Credit Risk Management whereby Markov Chains are used to "describe the probabilities that a certain company, country, etc. will either remain in their current state, or transition into a new state."[3] Similarly, Markov Chains can be used to predict future stock market conditions; while the stock market is volatile by nature, an approach using Markov Chains can be used to give greater insight into financial trends.[3]

2 Referenced Definitions and Theorems

This section outlines major definitions and theorems referenced throughout the rest of the paper.

2.1 Definitions

Definition 1. *Web Graph*[4]
A web graph is directed graph representation of interconnected states such that individual states are represented as nodes and possible ordered changes in state are represented as directed edges from one node to the next according to a probabilistic model.

Definition 2. *Transition/Stochastic Matrix*[4],[5],[6],[7]
An $n \times n$ stochastic matrix P contains entries representing the probability that a change in state occurs from one state to another. Specifically, the P_{ij} entry of the matrix, the entry in the i^{th} row and the j^{th} column, determines the probability that an input will change to state i given that the input starts at state j. In other words, "p_{ij} is the conditional probability that we are on the i-th page at step $n+1$ given that we were on the j-th page at step n", with p_{ij} equivalent to the P_{ij} entry.[6] Thus, each term in matrix is greater than or equal to 0, and in a normal transition matrix, the sum of the terms in each column add to 1 as the rows of the matrix construct the sample space for the possible change of states (and the probabilities over a sample space sum to 1).

Definition 3. *Steady-State/Equilibrium Vector*[5],[8] The steady-state or equilibrium vector is the vector such that

$$P\vec{x} = \vec{x}$$

which is the eigenvector corresponding to an eigenvalue of 1 for the regular stochastic matrix P. This vector $\vec{x}$ represents the convergence of the probabilities upon repeated multiplications (tending towards infinity) by P given any initial state x_0.

Definition 4. *Dangling Node*[4]
A dangling node exists in a web-graph when the matrix representation of the graph has a column of all 0s. This implies that the node does not allow a following state as no outgoing links exist.

Definition 5. *Altered Transition Matrix*[4]
In the case that a dangling node exists, it is not initially possible to construct a regular stochastic matrix from the given probabilities. To instead construct a representation of the probabilities, we replace the column of 0s with a column of the form

$$\begin{bmatrix} \frac{1}{N} \\ \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{bmatrix}$$

where N is the number of nodes in the graph. This representation defines that the following node will be picked randomly from the N possible nodes and is referred to as an altered transition matrix.

Definition 6. *Irreducible Graph*[4] An irreducible graph contains distinct nodes that can pairwise be reached through traversal of the web-graph. Specifically, "a graph is called irreducible if for any pair of distinct nodes, we can start from one of them, follow the links in the web graph, and arrive at the other node, and vice versa."[4]

Definition 7. *Reducible Graph*[4]
According to Shum, a reducible graph is any graph that is "not irreducible" as defined above.[4] A reducible graph forces the issue that the probabilities of arriving at a specific node are altered by the creation of a cycle within a subset of the web graph, preventing traversal to previous nodes despite their nonzero reachability.

Definition 8. *Damping Factor*[4],[6],[8]
A positive constant damping factor β chosen from $[0, 1]$ is utilized for reducible graphs such that we replace the transition matrix P with $\widetilde{P}$ defining

$$\widetilde{P} = \beta P + \frac{1-\beta}{n} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

where we multiply the $n \times n$ matrix of ones by the factor $\frac{1-\beta}{n}$, following Shum's definition (which is equivalent to, but expressed differently than Rousseau's and Tanase's, et al. representations).[4,6,8] Rousseau expands on the explanation of reducible graphs as characterized by multiple characteristic polynomial roots of $\det(\lambda I - A) = 0$ such that $\lambda = 1$ or possessing "other eigenvalues with modulus equal to 1".[6]

Definition 9. *Markov Chain*[1],[6]
A Markov chain, or a Markov chain process, is the iterative method of constructing a sequence of independent future state prediction data given initial input in the form of an n-term state vector $\vec{x}_0$ and an $n \times n$ stochastic matrix P. Each prediction $\vec{x}_{n+1}$, given $\vec{x}_n$, is found through the matrix-vector multiplication $A\vec{x}_n$. Given a stochastic matrix, it is possible to create an altered or dampened stochastic matrix such that the adjusted matrix is regular.

2.2 Theorems

Theorem 1. *Perron-Frobenius Theorem*[8]
The Perron-Frobenius Theorem states that given a positive, column stochastic matrix M, the following three properties are satisfied:

1. 1 is an eigenvalue of multiplicity one for M.
2. 1 is the largest eigenvalue; all the other eigenvalues have absolute value smaller than 1.
3. The eigenvectors corresponding to the eigenvalue 1 have either only positive entries or only negative entries. In particular, for the eigenvalue 1, there exists a unique eigenvector with the sum of its entries equal to 1.

as stated in Tanase, et al.[8]

Theorem 2. *Power Method Convergence Theorem*[8]
The Power Method Convergence Theorem states that given a positive, column stochastic $n \times n$ matrix M, we denote $\vec{v}^*$ as its probabilistic eigenvector corresponding to the eigenvalue 1. If we then take $\vec{z}$ as a column vector with all entries equal to $\frac{1}{n}$, the sequence $\vec{z}, M\vec{z}, \ldots, M^k\vec{z}$ converges to the vector $\vec{v}^*$. The above definition was stated in Tanase, et al.[8]

3 PageRank Algorithm

This section describes the basis for the stochastic model characterized by standard PageRank algorithms. In addition, the material in this section describes our implementation of these techniques, encapsulated in our Python function definition $rank(ranks)$, which takes in as input a list of links emanating from each node represented as a 2-dimensional list, and outputs a single, ordered list of rankings. This algorithm will focus largely on the computation and construction of the Markov chain algorithm for PageRank. For a more probabilistic approach refer to Rousseau's consideration of conditional probabilities and their effects.[6]

3.1 Adjacency Matrix Construction and Dangling Nodes

Given an adjacency list, effectively an array with a list of connected nodes, we must construct a transition matrix to represent the web graph that the links define. In a simple PageRank algorithm, given a set of directed edges emanating from a single node, we define the probability of choosing any one of the emanating edges to be equal to the probability of choosing any other emanating edge. Therefore, for a basic case with at least one directed edge from each node, we guarantee that the sum of the entries in any column equals 1 and that all the entries are non-negative. Thus, following Shum's algorithm example, we define an $n \times n$ matrix P with entries in row i and column j as

$$P_{ij} = \begin{cases} \frac{1}{C(j)} & \text{if there is a link from } j \text{ to } i \\ 0 & \text{in all other cases} \end{cases}$$

where we define $C(j)$ to be the number of total outgoing edges from node j.[4] However, we come upon an issue in the case of a dangling node. If a dangling node exists in an adjacency list, it is impossible to create a normal stochastic matrix without alteration, as a column of 0s does not allow for the formation of a normal stochastic matrix as the sum of the elements in the column never equals 1. Thus, again following Shum's example, in the case that we come across columns of 0s in our transition matrix P representation, we replace the column of 0s with a column of the form $\begin{bmatrix} \frac{1}{n} \\ \frac{1}{n} \\ \vdots \\ \frac{1}{n} \end{bmatrix}$ so as to define that a random node will be picked after reaching a node with no emanating edges.[4] We define this matrix to have entries of the form

$$P_{ij} = \begin{cases} \frac{1}{C(j)} & \text{if there is a link from } j \text{ to } i \\ \frac{1}{n} & \text{if there are no emanating links from node } j \\ 0 & \text{in all other cases} \end{cases}$$

with $C(j)$ and n as defined previously.[4]

Now, let us consider the implementation of this algorithmic idea in our code. First, we must initialize the size of the matrix to be $n \times n$, where n is the number of nodes. We access this by taking the length of the input vector *ranks*, which has a list of connected nodes for every node, so it has n elements. Now, we case on whether each node is a dangling node or not. If the node is dangling, we set each element in the column to have probability $\frac{1}{n}$. Otherwise, we construct each element of the matrix to have probability $\frac{1}{C(j)}$ where $C(j)$ is the number of elements that node j is directed to. From this, we use the numpy matrix function to construct the matrix given these values as entries for future compatibility with numpy.

```
#=========================================
#(1) CREATE MATRIX REPRESENTING PAGE LINKS
#=========================================

#initialize nxn matrix full of zeros
M = [[0 for i in range(0, len(ranks))] for j in range(0, len(ranks))];

#also need to transpose the given matrix
for i in range(0, len(ranks)):
  #if a node has no outgoing connections,
  #assign equal probability to go to every other node (1/n)
  if len(ranks[i]) == 0:
    for j in range(0, len(ranks)):
      M[j][i] = 1.0 / len(ranks);
  else:
    for j in range(0, len(ranks[i])):
      M[ranks[i][j]][i] = 1.0 / len(ranks[i]); #probability from one node to all other nodes

M = np.matrix(M);
```

3.2 General Solution

Our generic solution then uses the Power Method Convergence Theorem to compute a close approximation to the steady-state vector that defines the ranks. As repeated multiplications by the matrix P to an n-valued vector with terms equal to 1 approaches an eigenvector for $\lambda = 1$, we see that a steady-state vector is approached, which is our goal. This resultant steady-state vector then defines the output rankings based on its elements.

In code, this process is written as a loop iterating through repeated multiplications and terminating when equality is achieved, or after 1000 iterations of the loop are completed, which, as shown above, is likely to be effective for sparse matrices, as these matrices quickly converge on multiplication by a defining vector, as Tanase et al.[8] A limit of 1000 iterations was chosen to limit runtime while still providing precise output.

```
#==================
#(2) DO POWERMETHOD
#==================

v = np.array([[1 for i in range(0, len(ranks))]]).transpose();

#iterate until convergence or until 1000 iterations (so it doesn't take too long)
numIterations = 0
while(not np.array_equal(np.dot(M, v), v) and numIterations < 1000):
  v = np.dot(M, v);
  numIterations = numIterations + 1;
```

Following this process, we clean our resultant vector such that the indices (and thus, nodes) are ranked in descending order, with the highest ranked node occurring first in the output vector and the lowest ranked node occurring last. Finally, we return the resulting output, our ranked list of nodes.

```
#==========================================================
#(3) CLEAN UP THE OUTPUT TO A SORTED (by index) PYTHON LIST
#==========================================================

#final vector is the result
sort = np.argsort(v.flatten()); #sort in ascending order
sort = list(reversed(sort.tolist()[0])); #reverse to descending order

#fin.
return sort;
```

3.3 Damping Factors

In the case that our stochastic matrix P is reducible, we must add an additional step to ensure continued reachability for initial nodes. We characterize this by calculating a secondary positive stochastic matrix. To do so, we utilize a damping constant β set between $[0, 1]$ as a coefficient regarding our matrix multiplication as follows:

$$M = \beta P + \frac{1-\beta}{n}\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

This also solves issues regarding repeated characteristic polynomial roots for $\lambda = 1$ or the possibility of the presence of multiple eigenvalues with modulus equal to 1, as Rousseau notes.[6] We may also use this technique of damping in general, not just on reducible graphs to incorporate randomness into the probabilistic model.[8]

In our algorithm, we follow this process exactly, defining an all-ones matrix, a damping factor, and then redefining our matrix M with the damping factor specified according to the equation above. From this, we proceed to the general solution as defined previously in Section 3.2.

```
#------------------------------
#(1A) HANDLING REDUCIBLE GRAPHS
#------------------------------

#convert to a numpy matrix
O = np.matrix([[1 for i in range(0, len(ranks))] for j in range(0, len(ranks))]); # nxn matrix full of ones
d = 0.85 #standard value for damping factor, (will change answers for test case)
M = d * M + (1.0 - d) / len(M) * O; #dampen M
```

4 Example

Let us consider a working example and defining behavior for the Markov chain model implemented in the previous section. This example will touch on some of the major points defined, first calculating the expected result, and then describing intermediate and returned results from the algorithm.

Let us consider the following web graph with a damping factor of 1:

$$\begin{tikzcd} A \arrow[r, leftrightarrow] \arrow[d] & B \\ C & D \arrow[l] \arrow[u] \end{tikzcd}$$

We input the adjacency list of connected edges of the form adjList $= [[1,2],[0],[],[1,2]]$ into our algorithm, with 0 (and index 0) corresponding to node A, 1 (and index 1) corresponding to node B, 2 (and index 2) corresponding to node C, and 3 (and index 3) corresponding to node D. The algorithm output will be described following the expected result by computation.

Considering the computation by hand, from this representation, we may construct the adjacency matrix P as follows:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ .5 & 0 & 0 & .5 \\ .5 & 0 & 0 & .5 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

It is easily observable that the adjacency matrix P has a dangling node, as represented by the column of 0s. Thus, we construct the altered transition matrix:

$$\widetilde{P} = \begin{bmatrix} 0 & 1 & .25 & 0 \\ .5 & 0 & .25 & .5 \\ .5 & 0 & .25 & .5 \\ 0 & 0 & .25 & 0 \end{bmatrix}$$

By calculation, a resulting eigenvector with a column sum equal to 1 for $\lambda = 1$ is

$$\vec{r} = \begin{bmatrix} r(A) \\ r(B) \\ r(C) \\ r(D) \end{bmatrix} = \begin{bmatrix} .357 \\ .286 \\ .286 \\ .071 \end{bmatrix}$$

Thus, the expected output is $A > B \geq C > D$, with B and C possibly switched in the output ordering based on their equivalent value.

Now, returning to our algorithm, intermediately, in constructing an eigenvector in our algorithm, we observe that

$$\vec{v} = \begin{bmatrix} 1.42857143 \\ 1.14285714 \\ 1.14285714 \\ 0.28571429 \end{bmatrix}$$

Now, observe that

$$\frac{1}{1.42857143 + 1.14285714 + 1.14285714 + 0.28571429}(\vec{v}) = \begin{bmatrix} .357 \\ .286 \\ .286 \\ .071 \end{bmatrix} = \begin{bmatrix} r(A) \\ r(C) \\ r(B) \\ r(D) \end{bmatrix}$$

Indeed, as expected, our algorithm returns as output $[0, 2, 1, 3]$, meaning that A has the highest rank, C has the second highest rank, B has the second lowest rank, and that D has the lowest rank. Thus, our output matches the expected value apart from the ordering of B and C, which is accounted for and noted in the calculation above.

5 Conclusion

The applications to Markov Chains and the PageRank algorithm span many fields, from Biology to Computer Science. While the implementation for PageRank described here is simple, it's efficacy demonstrates the immense power of these algorithms. While Google (and other large search engines) have moved away from such a simple model of PageRank, their underlying technology is still identical to what is demonstrated here: namely, using Markov Chains for predicting most relevant pages. In fact, this technique can be explored further. For example, a current open research question is Hidden Markov Models, which cluster probability distributions through processes assumed to be Markov chains, but have hidden states, preventing definitive conclusions on the internal ranking of variables.[9] This predictive model will most definitely have a plethora of applications for many years to come.

References

[1] Weisstein EW. Markov Chain. Wolfram MathWorld. http://mathworld.wolfram.com/MarkovChain.html. Accessed December 14, 2018.

[2] The History of PageRank and Iterative Searching Algorithms. Cornell University. https://blogs.cornell.edu/info2040/2017/10/25/the-history-of-pagerank-and-iterative-searching-algorithms/. Published October 25, 2017. Accessed December 14, 2018.

[3] Myers DS, Wallin L, Wikström P. *An Introduction to Markov Chains and Their Applications Within Finance.*

[4] Shum, K. (2013). *Notes on Page Rank Algorithm* (tech.). *Notes on Page Rank Algorithm.* `http://home.ie.cuhk.edu.hk/~wkshum/papers/pagerank.pdf`. Accessed 13 December 2018.

[5] Lay, D. C., Lay, S. R., & McDonald, J. (2016). *Linear algebra and its applications.* Boston: Pearson.

[6] Rousseau, C. (2017, February 7). How Google works: Markov chains and eigenvalues. *Klein Project Blog.* `http://blog.kleinproject.org/?p=280`. Accessed 13 December 2018.

[7] *Stochastic Matrix.* Wolfram MathWorld. `http://mathworld.wolfram.com/StochasticMatrix.html`. Accessed 13 December 2018.

[8] Tanase, R., & Radu, R. (2009). Lecture #3: PageRank Algorithm - The Mathematics of Google Search. *Cornell University: College of Arts and Sciences.* Cornell University. `http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html`. Accessed 13 December 2018.

[9] Jurafsky, D., & Martin, J. H. (2011). *Hidden Markov Models* (tech.). *Hidden Markov Models.*