Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.3800 Introduction to Inference
Fall 2023

Computational Lab V

Issued: Monday, November 27 **Due:** Friday, December 8, 10pm

Wikipedia Page Ranking

You are going to rank Wikipedia pages according to a version of the celebrated PageRank algorithm developed by Google to order pages in its search results. Wikipedia pages, similar to other web pages, include hyperlinks to other Wikipedia pages, which can be represented using a directed graph, with an edge from page `A` to page `B` if there is a link to page `B` on page `A`.

To develop PageRank's notion of ranking of Wikipedia pages, we imagine someone surfing Wikipedia, randomly following links on each page she visits, which corresponds to a random walk on the directed graph. To make it more realistic, we assign a weight to each page that is inversely proportional to length of the title of that page, with the notion that our web surfer is more likely to follow a link to a page with a shorter name. In particular, if a page has d links on it, to pages with with weights $e_1, \ldots, e_d$, the probability of following link j will be $e_j / \sum_{j'=1}^{d} e_{j'}$.

Since this random walk corresponds to a homogeneous Markov chain whose states are the Wikipedia pages, we use as the scores (i.e., rankings) the steady-state probabilities for the states in the chain. As such, the higher ranked pages are those that are visited more frequently by our Wikipedia surfer.

As the entirety of Wikipedia is rather large, we will restrict our investigation to subsets of it. Even then, the state space is very large, and thus directly solving the detailed balance equations to obtain the stationary distribution is computationally prohibitive. Instead, we will rely on Monte Carlo methods to estimate this distribution.

Code and Data

The file `markov_chain.zip` contains the functions:

`markov_chain.py`	Your code goes here. Includes incomplete functions that you will fill in. Read the comments in this file to understand what code you have to add.
`util.py`	Python functions/classes that will be useful as you write your code. Please do not modify.
`stat.gml`	Provides a weighted directed graph of Wikipedia articles related to statistics, using the Python library `networkx`. Note that you will not need to modify the edge weights on this graph, but you will need to use the utility `normalize` to normalize distributions you use.

and the data:

`example_data.gml`	Provides example data for debugging purposes. It comprises a much smaller subset of Wikipedia of pages related to the topic of information theory.

You can run the code via:

```
python3 markov_chain.py <data_file> <samples> <iterations_between_samples>
```

Important: Do not change the function definitions in the code as they must remain consistent for automatic grading. The only parts of the code that you need to fill in are in blocks denoted by "YOUR CODE GOES HERE". Feel free to create additional functions as needed to help with the computation or to answer the questions in the lab.

(a) Fill in the function `approx_markov_chain_steady_state` in `markov_chain.py`. Your implementation should simulate the following variant of the Markov chain, starting at an arbitrary (e.g., random) initial state. From any particular state, with probability 0.9 the chain should follow the transition distribution of the original chain described above, and with probability 0.1 it should transition to a randomly chosen state.[1] Additionally, your function should take a sample once every `iterations_between_samples` iterations. The function should return a `Distribution` object that represents the empirical distribution over the samples it took. You will find it helpful to use the functions of the `Distribution` class. The transition probability for a node with out-degree[2] 0 is degenerate, so following the transition probability distribution will entail not changing the state.

Test your answer on `example_data.gml` with 2500 samples taken 1000 iterations apart. Repeat this twice and compute the divergence between the two distributions you obtain (in bits). In your writeup, include a plot of divergence between the two distribution (the two different empirical distributions when running the Markov Chain twice) when using 128, 256, 512, 1024, 2048, 4096, and 8192 samples, while keeping the number of iterations between samples to 1000. This may take a few minutes to run.

(b) Run your algorithm on the Wikipedia data in `stat.gml`. Collect 25 000 samples taken 1000 iterations apart. Repeat this 3 times. Determine the pages that

[1] This represents a model in which 10% of the time our Wikipedia surfer picks a Wikipedia page at random instead of following one of the links on the page she is currently viewing.

[2] The out-degree of a node is the number of directed edges emanating from it. The in-degree of a node is the number of directed edges arriving at it.

appear consistently among the 20 highest-ranked ones. This is a substantially longer computation. We recommend running this remotely, on `athena` for instance. To run it remotely, helpful commands include `ssh`, `rsync` (or `scp`), and `screen`.

(c) Now consider using this ranking methodology for web pages more generally, i.e., beyond just Wikipedia. There is no coding in this part.

(i) Should we expect the original Markov chain to be fully communicating? Explain.

(ii) Now consider the variant of the Markov chain in part (a), whereby with probability 0.1 we jump to a random web page instead of following one of the hyperlinks on the current web page. Is this modified chain fully communicating? Explain how the randomization affects the steady-state behavior of the chain.

(iii) What steps might one take to increase the ranking of a web page?

(d) A simple alternative to our PageRank-based methodology is to use as a page's score the degree (in-degree and out-degree combined) of its node in the graph. In your writeup, include a scatter plot of the Wikipedia pages' PageRank scores vs. their node-degree scores. Discuss the differences you observe. Why might a page have a high PageRank score but a comparatively low degree?

What to hand in: Upload to Gradescope your writeup as a single PDF file and completed file `markov_chain.py`. The writeup should include your answers for all parts. Scans/images of handwritten work are fine but please write neatly — we can't grade what we can't decipher!

Gradescope will run a number of tests which are visible to you. These are to make sure your code runs correctly in the Gradescope system. Please make sure you pass these tests as we will use the autograder to check your submission.