CMS/CS/Ec/EE 144 **Gurus: Sidd and Allison**

HW 4: PageRank & Centrality

Assigned: 01/29/2026 Due: 02/05/2026 by 11:59 PM (Midnight) PT

Collaboration is allowed for all problems and at the top of your homework sheet, please list all the people with whom you discussed. Crediting help from other classmates will not take away any credit from you. The details of the collaboration policy for this course are available in the Resources tab on Piazza.

You must turn in your homework electronically via Gradescope. ***Be sure to submit your homework as a single file.***

Start early and come to office hours with your questions! We also encourage you to post your questions on Piazza, as well as answer the questions asked by others on Piazza.

Coding + Data Analysis [70 points]

---

1 Approximately Central [30 points]

For large scale graphs that contains a million or even more nodes, some familiar centrality measures, such as betweenness centrality or closeness centrality, are prohibitively expensive to compute. For example, calculating the closeness centrality on a network of 24 million nodes using standard computing resources takes an astonishing 44,222 hours, or around 5 years, to compute. Even computing pagerank can be a difficult task in many cases.

Therefore, it is often important to compute centrality using *approximated methods*. In this problem, you will try to use *machine learning* to train a model that approximates betweenness centrality on real datasets. We consider a situation with small training networks (potentially subgraphs) and a large target graph. Our objective is to train a neural network on the small graph and evaluate it on the large graph.

Specifically, we consider a set of p2p networks using Gnutella. The dataset we use consisting of 9 networks ranging from 6,300 to 63,000 nodes. In order to train the model, we have to preprocess the dataset first. We use a Struc2Vec node embedding technique to represent the information of the graphs using generated graph vectors. Then the model can be trained using graph vectors and their corresponding ground truth.

We recommend you to use Google Colab to solve this problem. We also provide a Colab notebook **(here is the link)** to give you a step-by-step guideline. To run this notebook, you need to copy it into your own google drive and upload the datasets we provided to your drive as well. Specific instructions are included in the notebook.

Your task:

- The provided notebook includes a fully functional model training pipeline: preprocessing graph data, embedding the graph information and training the model. **We don't require you to modify any given function** for this problem. After the training, the pipeline computes a Kendall-tau similarity

score between the predicted centrality scores and their ground-truth (the exact betweenness function). We expect you to get a Kendall-tau score at least 0.7. In order to achieve this objective, you need to choose four pre-defined parameters wisely. We hope you can get familiar with this training pipeline by playing around the parameters. Turn in a description of your evaluations, including the best parameters you found and any connection you see to graph structure. *(This description can be a brief paragraph on what you noticed with varying some subset of the following: embedding size, # of layers, # of folds for cross-validation, # of epochs, etc.)*

- Evaluate your trained model on a larger graph. In this exercise, you will need to **write your own code** to evaluate your trained model on p2p-Gnutella04 dataset. We provide some hints in the Colab notebook. Report the Kendall-tau score you get for the evaluation.

Note: Please convert your Colab notebook into pdf when submitting it. You can include your findings and other answers in the notebook as well.

2 Bernoulli: Your Own Caltech Search Engine [40 points]

In this problem, you will build a working search engine for the Caltech domain that uses PageRank to improve search results. This will give you hands-on experience with the PageRank algorithm we've studied in class, and let you see how it performs on a real web graph.

You will implement the core PageRank algorithm, then use it to rank search results alongside traditional text-based relevance scoring. The starter code provides a complete web crawler, indexer, and search interface. All you need to do is implement the PageRank computation itself to help rank search results for relevance.

Your task:

(a) **[20 points]** Implement the `compute_pagerank()` function in `src/pagerank.py`. Your implementation should:

- Take as input a web graph (represented as a dictionary mapping URLs to lists of outgoing links) and the damping factor α
- Return a dictionary mapping each URL to its PageRank score
- Use the iterative power method discussed in class:

$$\pi = \lim_{n\to\infty} \pi_0 G^n \text{ where } G = \alpha P + \frac{1-\alpha}{n}(\mathbf{1}_{n\times n})$$

- Handle pages with no outgoing links appropriately
- Converge to a reasonable tolerance (e.g., when the maximum change in any PageRank score is less than 10^{-7})

The function signature and detailed specifications are provided in the starter code. Once implemented, run `compute_pagerank.py` to compute scores for your crawled web graph.

(b) **[20 points]** Use your search engine! First, crawl the Caltech domain by running `crawl.py`. Then compute PageRank scores by running `compute_pagerank.py` and perform at least 5 different searches using `search.py`. In your submission, include:

- A brief description of your crawling results (number of pages indexed, any interesting observations)
- The top 10 pages by PageRank from your crawl
- Your 5 search queries and a discussion of the quality of results for each query. Do the results make sense? How does PageRank affect the ranking compared to pure text relevance?

Submission instructions: Submit your completed `src/pagerank.py` file along with a brief write-up (PDF) containing the items requested in part (b). See the README.md in the starter code for detailed setup instructions and troubleshooting tips.

Extra Credit (up to 10 points): Implement a creative improvement to the search engine! This could be a custom ranking algorithm, a web interface, query expansion features, performance optimizations, or anything else you find interesting. Include a brief write-up explaining what you built and how it works. See the `README.md` for more ideas.

Theory [30 Points]

3 Pandemaniac Warm-Up [15 points]

In the coming week, you will be working on spreading an epidemic through a graph for Pandemaniac. You will do Pandemaniac in teams of 2 or 3 (no more than 4), so you should work to form your team this week and look out for the Piazza post asking for your group members and team name. In the meantime, this question is designed to help prepare you for this task.

In Pandemaniac, the epidemics you are working with will work as follows. Each round begins with the teams selecting some subset of the nodes on the graph. Each team is assigned a different color, and any nodes claimed by a team are assigned that team's color. Each node may be claimed by at most one team, but nodes can also start as unclaimed, in which case they remain uncolored. These uncolored nodes may then be convinced to adopt a color based on the colors of their neighbors. After they (or any nodes) adopt a color, they still may later be convinced to switch their color if their neighbors switch.

From the starting nodes, the epidemics spread iteratively. During each iteration, every node chooses its next color through a majority vote among itself and its direct neighbors. All colored direct neighbors of the node vote for their respective color with 1 vote; however, if the node itself is currently colored, it votes for its own color with 1.5 votes. If any color has a **strict majority** out of the total number of votes, then that becomes the next color for the node.

If there is no strict majority, then the node keeps its current color or remains uncolored. These cases are demonstrated in Figures 1 and 2. Note that it is (slightly) harder to convert a node after it has been acquired, but nodes can switch colors multiple times.

Your task: Is it necessary that a graph's epidemic colors always converge or stabilize? If yes, prove that this must always be true. Otherwise, give a counterexample in the form of a graph and initial coloring that do not converge.

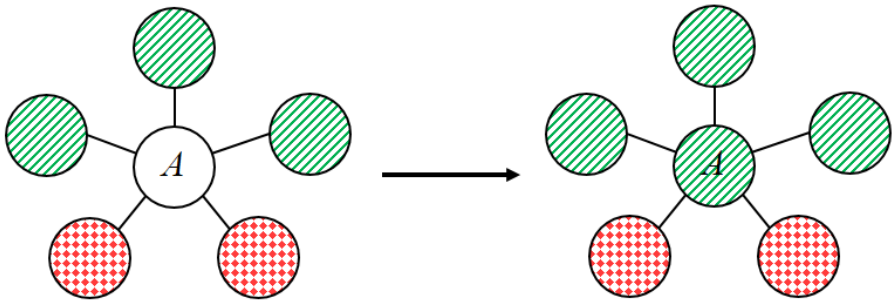

Figure 1: An uncolored node A converting to "striped" (since "spotted" gets 2 votes and "striped" gets 3 votes, which is a majority among the 5 votes).

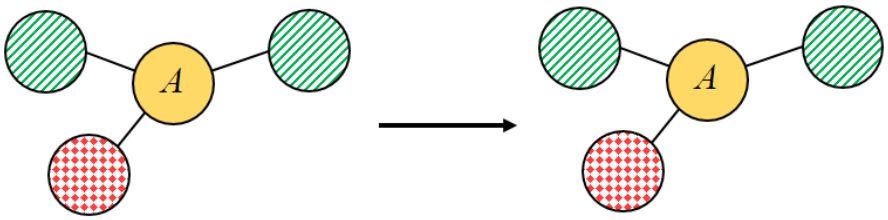

Figure 2: Node A doesn't change color because no color got the majority of votes ("striped" gets 2 votes, "dotted" gets 1 vote, "solid" gets 1.5 votes, none of them gets a majority among the 4.5 votes).

4 Warmup with PageRank and stationary distributions [5 points]

In class, we saw that PageRank can be viewed as calculating the stationary distribution of a transition matrix defined by a graph. In this problem we'll investigate the differences between various ways of calculating the stationary distribution. In particular, we have seen two ways of calculating the stationary distribution of a transition matrix P. One is the iterative method $\pi = \lim_{n\to\infty} \pi_0 P^n$ for some initial π_0, and the other one is to solve the equation $\pi = \pi P$.

For the following probability transition matrices, first use $\pi = \pi P$ to get the stationary distribution, and then show whether or not $\pi_0 P^n$ converges as $n \to \infty$. If it converges, show that $\pi = \lim_{n\to\infty} \pi_0 P^n$ does not depend on π_0 and interpret what this means about the stationary distribution. *Hint: you may want to diagonalize P to handle P^n easily. Feel free to use Mathematica/MATLAB to diagonalize P.*

(a) $\begin{pmatrix} 2/5 & 3/10 & 3/10 \\ 1/5 & 3/5 & 1/5 \\ 7/10 & 1/10 & 1/5 \end{pmatrix}$

(b) $\begin{pmatrix} 0 & 5/8 & 0 & 3/8 \\ 1 & 0 & 0 & 0 \\ 0 & 3/8 & 0 & 5/8 \\ 3/4 & 0 & 1/4 & 0 \end{pmatrix}$

5 Training to be a farmer [10 points]

A typical way to raise the PageRank of a page is to use "link farms", i.e., a collection of "fake" pages that point to yours in order to improve its PageRank. Our goal in this problem is to do a little analysis of the design of link farms, and how their structure affects PageRank calculations.

Consider the web graph. It contains n pages, labeled 1 through n. Of course, n is very large. As mentioned in class, we use the notation $G = \alpha P + \frac{1-\alpha}{n}(\mathbf{1}_{n\times n})$ for the transition matrix. Let r_i denote the PageRank of page i, and $r = (r_1, r_2, \cdots, r_n)$ denote the vector of PageRanks of all pages.

Note: For a page that has k outgoing links, we put $1/k$ for the corresponding entries of P. However, when a webpage has no outgoing links, we add a 1 as the corresponding diagonal element of P for making its row-sum one. Note that this makes G a valid transition probability matrix.

(a) You now create a new web page X (thus adding a node to the web graph). X has neither in-links, nor out-links. Let $\tilde{r} = (\tilde{r}_1, \tilde{r}_2, \cdots, \tilde{r}_n)$ denote the vector of new PageRanks of of the n old web pages, and

x denote the new PageRank of page X. In other words, $(\tilde{r}_1, \tilde{r}_2, \cdots \tilde{r}_n, x)$ is the PageRank vector of the new web graph.

Write $\tilde{r}$ and x in terms of r. Comment on how the PageRanks of the older pages changed due to the addition of the new page (remember n is a very large number). *Hint: Use the stationary equations to calculate PageRank, not the iterative approach.*

(b) Unsatisfied with the PageRank of your page X, you create another page Y (with no in-links) that links to X. What are the PageRanks of all the $n + 2$ pages now? Does the PageRank of X improve?

(c) Still unsatisfied, you create a third page Z. How should you set up the links on your three pages so as to maximize the PageRank of X? Justify your answer.