MATH/CS/STAT:4740 Large Data Analysis

Implementation of Google's PageRank algorithm in the uiowa.edu domain

PROJECT ADVISOR

David Stewart

GROUP MEMBERS

Madison Wang

Yujing Lu

Harsha Pitawela

Table of Contents

Background

When you search for a word in a word file, it gives you all the sentences that contain this word. But what if you are dealing with millions of files and just want to get one sentence out of them? That would be like searching a drop of water in the ocean. It is the same thing for web-searching. Imagine searching for a webpage that contains a specific word, how are you going to get the most relevant webpage out of billions of all webpages on the Internet which include that word? There needs to be a method to indicate you how relevant a webpage is to your searched word, so that you are more likely to find those highly relevant webpages in the top of your search results. That is what PageRank is meant to do.

PageRank (PR) is an algorithm designed by Google to rank webpages in the search results. According to Google, the intuition of the PageRank algorithm is to determine the importance of a webpage by counting the number of other webpages that point to it. The underlying assumption is that more important webpages are likely to be pointed to from more other webpages. After those numbers are stored, they are used to create matrices. Then after updating the matrix according the PageRank formula, each webpage should have its own importance index, or in other words, its own power. The final search result is ranked by webpages' powers in descending order, so that the webpage with highest powers would show up on the top due to their higher probabilities to be more relevant to the searched word.

Summary

For this project, our objective was to enable users to find pages in the uiowa.edu domain that contained the searched word. We wanted to list those pages in the order of their PageRank results.

Our PageRank project started with web crawling using tools like Beautifulsoup, HTTrack and Scraper etc. to scrape the webpages. After getting all URL addresses, their page contents were analyzed to extract the top 9 words in each webpage, which means the most frequent 9 words that show up in each webpage. Then we stored all the information into CSV files. The most important part was implementing the ranking algorithm. The PageRank algorithm outputted powers of URL's used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. The final goal of this project was to let the users search some key words and come up with a series of links that are highly correlated to the searched word, then order them in descending order using the probabilities calculated above.

Our project was implementing Google's PageRank with a smaller number of pages. We only scraped the pages end up with "uiowa.com". Since we had a relatively smaller size of data, we did not have to use other outside database to store our data.

Technologies Used

We mainly used Python 3.6 as our coding language for the project. As for the development environment, we used Anaconda's built-in notebook tool, Jupyter Notebook, which facilitated quick code execution and results analysis.

Some of the Python libraries we used include:

- Beautifulsoup - Web crawling
- Pandas, numpy, scikit-learn, nltk - Data preprocessing

Initial Data Collection

Primarily the project needed to identify the links between each URL's in the www.uiowa.edu domain. Hence a web crawler was built to initiate from www.uiowa.edu and crawl through each URL's that are linked to. This crawling continued recursively until all links among URL's in uiowa.edu domain were identified. We also avoided links directing to different sections of the same page, which are usually starting with a '#', as they would interfere with identification of unique URL's. The beautifulsoup library in python became handy in the crawling task.

While crawling through links, 9 most common words in each page were identified and recorded. In extracting these words, special attention was given to avoiding stop words such as is, are, of etc., as they would not add any meaningful significance to a page. Furthermore, only the root words were recorded in order to have a consistent set of words for each URL. For example, "appli" were recorded as the root word of "application". Python's nltk (natural language tool kit) library was very helpful in performing these filtering tasks.

At the end of initial data collection, we generated two CSV files in the following formats:

id	**url**	**word1**	**word2**	**…**	**…**	**word9**
1	http://www.uiowa.edu	iowa	univers			support

Table 1. webpage.csv

id	**From_url**	**To_url**
1	http://www.uiowa.edu	https://admissions.uiowa.edu/apply/

Table 2. link.csv

Data Preprocessing

In order to prepare matrices for the page rank calculation, we had to prepare data-frames from the above link.csv file. Techniques such as pivoting, aligning, and normalizing in pandas, and scikit-learn libraries were heavily used in this step. A data-frame was created from link.csv in the following format, which was later converted to a sparse matrix A in the PageRank formula. In this data-frame, 0 meant that there was no link existed between the corresponding From_url and To_url, whereas 1 meant that there was a link points to To_url from the corresponding From_url.

To_url / **From_url**	**http://www.uiowa.edu**	**https://admissions.uiowa.edu/apply/**	...
http://www.uiowa.edu	0	1	...
https://admissions.uiowa.edu/apply/	1	0	...
….	...	...	…

Table 3. From_url and To_url data frame, later converted to a sparse matrix A

Since we had information about all the links among webpages, we used a simple sum function to check the number of outgoing links for each URL. From the table below, it was shown that http://admissions.uiowa.edu had 4 outgoing links, and http://admissions.uiowa.edu/ had 3 outgoing links, while http%3A/recserv.uiowa.edu and http://Miriam-thaggert@uiowa.edu had none, which made them "dead-end links". This makes sense because http%3A/recserv.uiowa.edu is personal reservation link and http://Miriam-thaggert@uiowa.edu is an email address. However, when there existed more dead-end links like these two in our URL list, how should we address this issue in the algorithm?

http%3A/recserv.uiowa.edu	0.0
http://Miriam-thaggert@uiowa.edu	0.0
http://admissions.uiowa.edu	4.0
http://admissions.uiowa.edu/	3.0
…	…

Table 4. The number of outgoing links for each URL

PageRank Calculation

To deal with dead end links, we decided to create an indicator matrix, which had value 0's for URL's that had outgoing links and value 1's for dead-end URL's. Including this indicator matrix D in the formula would allow us to locate the dead-end URL's and deal with them separately instead of ignoring them in the iterative computation.

TO_URL	max_value
http%3A/recserv.uiowa.edu	1.0
http://Miriam-thaggert@uiowa.edu	1.0
http://admissions.uiowa.edu	0.0
http://admissions.uiowa.edu/	0.0
…	…

$$\longrightarrow \begin{pmatrix} 1.0 \\ 1.0 \\ 0.0 \\ 0.0 \\ \dots \end{pmatrix}$$

Table 5. Inserting an indicator column max_value to indicate whether the URL is a dead end. 1 means it is, 0 means it is not. Then create an indicator matrix D based on the table.

We then used the following formula to calculate the powers or relative importance of URL's in the uiowa.edu domain:

$$(P^{t+1})^T = (P^t)^T A + ((P^t)^T D)\frac{e^T}{n}$$

where,

P^{t+1} = Probability matrix at (t+1)th iteration (shape 7349×1);

A = Sparse matrix created from data frame of (From_url, To_url) (shape 7349×7349);

D = Matrix where element is 1 if dead end and 0 otherwise (shape 7349×1);

e = Column matrix with 1 as the only element (shape 7349×1);

n = 7349 (Total number of URL's).

To_url / **From_url**	**http://www.uiowa.edu**	**https://admissions.uiowa.edu/apply/**	...
http://www.uiowa.edu	0	1	...
https://admissions.uiowa.edu/apply/	1	0	...
….	...	...	…

$$\Big\downarrow$$

$$\begin{pmatrix} 0 & 1 & \dots \\ 1 & 0 & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

Table 6. Use Table 3 to generate matrix A

Above calculations were performed using numpy arrays in Python's numpy library. After running the above calculation iteratively, we were able to use the final matrix P to come up with a table of URL's and their relative importance/PageRank indicated by the probabilities/powers. We saved this table into prob.csv file for later use.

TO_URL	**prob**
http%3A/recserv.uiowa.edu	0.00000191
http://admissions.uiowa.edu	0.00000541
…	…

Table 7. prob.csv is generated directly from the final matrix P result

Below are the two lists that present the first few URL’s and the last few URL’s in descending order of their probabilities.

https://admissions.uiowa.edu/apply/
http://www.uiowa.edu
https://iam.uiowa.edu/whitepages
https://www.uiowa.edu/students
https://now.uiowa.edu
https://now.uiowa.edu/media-relations

Table 8. URL’s with highest probabilities

http://hris.uiowa.edu/classcomp/merit/descr_docs/
https://education.uiowa.edu/person/
http://www2.education.uiowa.edu/directories/
https://belinblank.education.uiowa.edu/students/
https://transportation.uiowa.edu/
https://uc.uiowa.edu/file/

Table 9. URL’s with lowest probabilities

Further work

Since all our URL’s were unique, and there was a unique power for each of them, in order to match each URL with its corresponding power and its top 9 words, we chose to use dictionaries in Python to store all the information.

Recall in webpage.csv file, which contains URL’s and their corresponding ID’s, each URL’s also have their top 9 frequently shown words. We then created a dictionary called urldict using that information. In this dictionary, we used URL’s as dictionary keys, and lists of top 9 words as values. This is what urldict looks like:

{'https://uiowa.edu': ['iowa', 'univers', 'student', 'search', 'read', 'match', 'directori', 'first', 'support’], …}.

In previous steps, we also generated a prob.csv, which contains URL’s and their corresponding powers. By using the prob.csv, we created another dictionary called probdict, which has URL’s as keys and their corresponding powers as values:

{'http%3A/recserv.uiowa.edu': [1.91e-06], 'http://Miriam-thaggert@uiowa.edu’: [1.29e-06], …}.

Our last step was combining the above two dictionaries to create a nested dictionary called worddict. This worddict dictionary included all the information we needed to create a mini search engine for the uiowa.edu domain. In this dictionary, we used frequent words as keys, and sub-dictionaries as values. In each sub-dictionary, we used URL’s, which contained that specific word, as keys and the corresponding powers as values. This way, we could extract a list of URL’s along with their corresponding powers for a specific word. The nested dictionary worddict looks like this: {‘abw’: {'https://classrooms.uiowa.edu': [3.81e-05], 'https://classrooms.uiowa.edu/': [5.93e-06]}, …}

After all information was stored properly in the above three dictionaries, we wrote a while true loop to create an interactive surface, which would ask for an one-word input, and take it to search for this word in worddict, then return the sorted URL list with powers if that input word is included in our word list; and print “Sorry, we don't have that word in our word list” otherwise.

One example is shown below. When we search for the word “Student”, our program would give us:

https://admissions.uiowa.edu/apply/apply?utm_so...	0.573009
http://www.uiowa.edu	0.571104
https://iam.uiowa.edu/whitepages	0.380491
https://www.uiowa.edu/students	0.377793
https://uiowa.edu	0.193264
https://web.archive.org/web/20081116222732/http...	0.003706
https://web.archive.org/web/20080802013148/http...	0.003535

Table 10. The search result for word “Student”

Final takeaway

Since our mimic result only have 6,000 to 7,000 URLs, we can store the data in a simple matrix and our computer storage is enough for storing all of them. However, when it comes to Google, their approach is distributed storage (BigTable), or Amazon's (SimpleDB), etc. There is also an open source system called Hypertable, which can run on top of Apache Hadoop for instance.

Our simple version of search engine only allows us to search one at a time because we store the keywords and their URLs in a dictionary. In the real Google search engine, they can search the whole sentences with multiple keywords. Besides that, they can order each word to see what is the most central words.

Since our search engines are ordering corresponding URLs by the probability, it can only give the result of the most common word. In the real world, the Google search engine will search not only the most common word but also the most uncommon words. The restriction of the certain rage of URLs is disappearing for Google search engine. It will explore all the possible URLs.

The real word Google search engine will search the entire sentence with the highlight of the most important key word. Because the restriction of our searching database, every word we search we identified as a single key word. It would be hard to search if some phrases must be identified as a certain combination.

References

Enge, E. (2019, January 28). How does Google create search results? Learn here! Retrieved from https://www.stonetemple.com/how-googles-search-results-work-crawling-indexing-and-ranking/

Page Rank checker is a free service to check Google™ page rank instantly via online PR checker or by adding a PageRank checking button to your web pages. (n.d.). Retrieved from https://www.prchecker.info/check_page_rank.php

Project Github Repo: https://github.com/hpitawela/pagerank_uiowa_domain

Timsoulo. (2019, April 25). Google PageRank is NOT Dead: Why It Still Matters. Retrieved from https://ahrefs.com/blog/google-pagerank/