Assignment 2 – Web Crawler

Shilong Li
67893912
shilonl2@uci.edu

Wenjun Huang
36885180
wenjunh3@uci.edu

Yang Liu
88889586
yangl73@uci.edu

1 Introduction

In this assignment, we implemented a Web crawler that crawls the domains of UCI's ICS School and counts the word frequency.

1.1 Web Links Crawling

When doing the Web crawling, we sent HTTP request to the cache server and received the corresponding response. The response would be discarded if (1) it does not have a **200** status code; (2) the response has empty Web content. After the first processing, we built a DOM tree object based on the response content with the help of **lxml**[2] library. The links that can be accessed through the current web page were obtained by extracting the information of the "href" attribute in the HTML "a" tag (e.g., <a href='/path/to/site'>). We used XPath expression (i.e., document.$x("//a")) to collect all the values of the attribute and filtered out the empty strings.

There are five kinds of "href" values[1]:

- An absolute URL (e.g., href="http://www.example.com/default.htm").
- A relative URL (e.g., href="default.htm").
- Link to an element with a specified id within the page (e.g., href="#section2").
- Other protocols (e.g., ftp://, mailto:, file:).
- A script (e.g., href="javascript:alert('Hello');").

In this case, we must turn all "href" values into normalized URLs. Here are our strategies: (1) Adding the protocol of the current Web URL to those that do not have a protocol; (2) Adding protocol of domain of the current Web URL to those relative URLs; (3) Discarding fragment part; (4) Sorting parameter and query strings in terms of key-value pairs; (5) Decoding a rewritten URL by URL defense[4]; (6) Ignoring script and protocol other than "http" and "https". Every strategy contributed to our final results. Most importantly, (4) helped reduce a lot of crawling time since there were many URLs referring to the same page but with different orders of key-value pairs in the parameter/query strings (e.g., "a=b&c=d" and "c=d&a=b"). The number of mutations increases exponentially when there are various key-value pairs. Furthermore, it can avoid infinite traps or loops by eliminating the same pages. Actually, some corner cases must be taken into account such as "href='//www.a.c/path'" and "href='https://www.a.c/path;b'" during the implementation process.

1.2 Web Content Crawling

When crawling a web that contains the information we are interested in, the crawler extracted and analyzed the textual information with the help of the packages **bs4**[5], and **nltk**[3]. We refactored the HTML document to a nested data structure, then extracted all the text from the page. The extracted text was fed into a function for standardization after converting all uppercase characters into lowercase. In the standardization function, we first tokenized the text, then tagged the list of tokens with speech taggers based on nltk's recommendation. After assigning the speech taggers, we implemented special characters and tagger category checks. Specifically, the tokens that contain special characters or that belonged to the undeclared tagger category were removed. In the last stage before counting the meaningful words, the remaining tokens were lemmatized according to their taggers and filtered out the stop words. Since we want to avoid the sets of similar pages with no extra information, we implemented simhash[6] to compare the similarity between the current page and the pages we crawled. Only the page whose hamming distance with other pages is larger than the threshold (in our crawler, the threshold was set to 8) would be counted. Two counters, one counts the word frequency, and the other one counts the total word number of the page, were saved in the disk and were updated based on the token list when a qualified page was crawled. A list that stores the hash values of the qualified pages was also saved for calculating hamming distance. More details can be referred to in our code.

2 Q & A

2.1 How many unique pages did you find?

We did not count any web pages in the "gitlab.ics.uci.edu" domain since they do not include useful information except for code.

At the very beginning, we did not take the similarity of web pages into account. It took about five days to run the program and crawled **110432** unique pages, which was huge. When extracting the information from the logs, we found that there were many pages sharing similar web content. However, in fact, similar pages should be considered duplicates rather than independent ones. Hence, we came up with two ideas to eliminate the redundancy: (1) URL; (2) Web content. We did not adopt the (1) strategy since it mainly depends on the Web developers. The difference between URLs dominates the page similarity calculation.

Therefore, simhash and hamming distance strategies were applied to efficient crawling. If a page is similar to the pages that have been crawled, it would be discarded. So as the links in its Web content. Finally, we found **12208** unique pages. The detailed information is introduced in Section 1.2.

2.2 What is the longest page in terms of the number of words?

The longest page in terms of the number of words is https://www.ics.uci.edu/ kay/wordlist.txt. It is a text file that contains 380152 words.

2.3 What are the 50 most common words in the entire set of pages?

The 50 most common words in the entire set of pages are listed in Table 1. The result meets our expectations since the words shown in the table generally are related to ICS school (e.g., research, student, computer, etc.). In addition to the words related to ICS's research fields, other words are common words used in English, such as will, use, also, etc.

2.4 How many subdomains did you find in the "ics.uci.edu" domain?

The number of unique pages detected in each subdomain is listed in Table 2. The subdomains are ordered alphabetically, from the top to the bottom of the left column, then from the top to the bottom of the right column.

Word	Number	Word	Number
research	16371	page	5983
student	16070	current	5895
computer	14091	people	5822
will	13013	one	5750
use	11952	support	5749
science	11818	contact	5632
information	10910	year	5625
data	10757	value	5557
uci	9398	may	5534
can	9356	bren	5532
event	9070	search	5476
project	8944	design	5335
course	8929	graduate	5318
system	8852	compute	5281
news	8511	make	5264
ic	8440	university	5194
software	7706	irvine	5077
policy	7691	model	4968
school	7382	paper	4926
2022	7185	also	4577
work	6920	faculty	4539
time	6918	first	4530
program	6507	book	4499
ramesh	6429	update	4294
new	5988	group	4238

Table 1: 50 most common words in the entire set of pages.

3 Discussion

In this section, we discuss the troubles we encountered in this assignment. There are mainly two kinds of issues: network and text processing. Network issues include the number of HTTP requests to the cache server exceeding the maximum entry limit, the cache server being shut down, and an unstable VPN/ssh connection. After many attempts, we wrote a script to run the program in the background. Here we did not filter out the large files since we thought they might contain useful information. One main issue in text processing is word lemmatization. We set two requirements for text processing: (1). separating text based on semantics instead of simply based on space or special characters; (2). implementing lemmatization before count, e.g., “I’m” would be lemmatized to “I be”, and “runs” would be lemmatized to “run”, etc. However, lemmatization is difficult since no library returns lemmatized words directly. Therefore, we classified the words based on their parts of speech provided by wordnet and specified the classes that should be lemmatized. We used the library **nltk** to do lemmatization. And we did not find any web pages that had URLs like “today.uci.edu/department/information_computer_sciences/*”.

4 Conclusion

In the assignment, we implemented a crawler that crawls the domains of ICS school. We implemented a set of functions to enable the crawler to avoid infinite traps and decode a rewritten URL. Apart from crawling the pages, we also implemented functions to extract and analyze text. We used lemmatizer and simhash to increase the precision of the word count. We found 12208 unique pages in total. The longest page is https://www.ics.uci.edu/ kay/wordlist.txt, which contains 380152 words. The 50 most common words are shown in Table 1, which is consistent with our expectations. In addition, the subdomains found in the “ics.uci.edu” are listed in Table 2.

Subdomain	Number	Subdomain	Number
accessibility.ics.uci.edu	1	fr.ics.uci.edu	3
acoi.ics.uci.edu	69	futurehealth.ics.uci.edu	112
aiclub.ics.uci.edu	1	grape.ics.uci.edu	978
archive.ics.uci.edu	6	graphics.ics.uci.edu	3
asterix.ics.uci.edu	7	graphmod.ics.uci.edu	1
cbcl.ics.uci.edu	529	hack.ics.uci.edu	1
cert.ics.uci.edu	3	hai.ics.uci.edu	2
checkin.ics.uci.edu	5	helpdesk.ics.uci.edu	4
chenli.ics.uci.edu	9	hobbes.ics.uci.edu	1
cml.ics.uci.edu	150	hpi.ics.uci.edu	3
code.ics.uci.edu	13	hub.ics.uci.edu	3
computableplant.ics.uci.edu	29	i-sensorium.ics.uci.edu	1
containers.ics.uci.edu	1	iasl.ics.uci.edu	20
cradl.ics.uci.edu	20	industryshowcase.ics.uci.edu	20
create.ics.uci.edu	7	informatics.ics.uci.edu	1
cwicsocal18.ics.uci.edu	10	instdav.ics.uci.edu	1
cyberclub.ics.uci.edu	15	intranet.ics.uci.edu	10
dgillen.ics.uci.edu	23	ipubmed.ics.uci.edu	1
duttgroup.ics.uci.edu	77	isg.ics.uci.edu	182
dynamo.ics.uci.edu	1	jgarcia.ics.uci.edu	23
elms.ics.uci.edu	1	keys.ics.uci.edu	3
emj.ics.uci.edu	44	luci.ics.uci.edu	4
esl.ics.uci.edu	2	mailboss.ics.uci.edu	1
evoke.ics.uci.edu	7	malek.ics.uci.edu	1
flamingo.ics.uci.edu	12	mcs.ics.uci.edu	83

Table 2: The number of unique pages detected in each subdomain. The subdomains are ordered alphabetically, from the top to the bottom of the left column, then from the top to the bottom of the right column.

A Instructions

A.1 Create Conda Environment

Under the project directory, execute the following commands:

- conda create -n a2 python=3.10
- conda activate a2

A.2 Install Additional Libraries

Under the project directory, execute the following commands:

- pip install -r requirements.txt
- python
- > > > import nltk
- > > > nltk.download('punkt')
- > > > nltk.download('averaged_perceptron_tagger')
- > > > nltk.download('wordnet')

A.3 Run the Program

Under the project directory, execute the following command:

- nohup python launch.py > output.log 2>&1 &

This command enables our program running in the background.

A.4 Check the Status of the Program

You can use any of the following commands to check the status:

- cat output.log
- cat Logs/Worker.log
- tail -5 output.log
- tail -5 Logs/Worker.log
- ps aux | grep [pid]

A.5 Restart the Program

You can simply execute the following command to clean the previous data:

- chmod +x ./clean.sh && ./clean.sh

A.6 Obtain the Results

For Q1 and Q4, execute the command:

- python ./utils/count_distinct_url.py ./Logs/Worker.log

For Q2 and Q3, execute the command:

- python ./utils/extract_word_num.py f_path_1 f_path_2

, where f_path_1 and f_path_2 are the paths of **counter_all_word_num.pkl** and **counter_page_word_num.pkl**, respectively.

References

[1] HTML a href Attribute. https://www.w3schools.com/tags/att_a_href.asp#:~:text=Possible%20values%3A,htm%22). Accessed: February 2023.

[2] lxml - XML and HTML with Python. https://lxml.de/. Accessed: February 2023.

[3] Steven Bird, Ewan Klein, and Edward Loper. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.", 2009.

[4] Inc. Proofpoint. How do I decode a rewritten URL. https://help.proofpoint.com/Threat_Insight_Dashboard/Concepts/How_do_I_decode_a_rewritten_URL%3F. Accessed: February 2023.

[5] Leonard Richardson. Beautiful soup documentation. April, 2007.

[6] Caitlin Sadowski and Greg Levin. Simhash: Hash-based similarity detection, 2007.