Assignment 1 Web Crawler Report

CS 4250 Web Search and Recommender Systems

Devin Khun, Daniel Ho, Caden Minniefield, Tony Swank, and Thet Wai

1 Web Crawler

In our implementation of the web crawler, for libraries, we utilized Beautifulsoup, requests, urllib, csv, and lingua Python modules. lingua was used for language detection, in which we used a dictionary to have keys for each detected language. We used urllib in order to combine and parse the url of the links, and csv in order to read and write out to csv files. Beautifulsoup was used to coarse the html documents and extract links (a href). requests was used to provide a simple way to make http requests to get the content.

2 Tokenization And Stemming

2.1 Tokenization

We used BeautifulSoup to parse the text from HTML files, extracting only the readable content. After parsing, using Regular Expressions, we retained only alphanumeric characters and certain special characters such as @$&%, which might be important. All other symbols were removed, and line breaks such as \t and \n were replaced with spaces. Finally, we applied NLTK's tokenizer, which returned an array of tokenized words.

2.2 Stemming

The stemming was pretty simple in that we used the default implementation of the PorterStemmer provided by the Natural Language Toolkit to reduce to their basic forms and categorize them.

2.3 Alternate Considerations

Before implementing with NLTK, we tried to implement using Spacy, which was a newer and similar language processor to NLTK but after reading and testing it, it only supports lemmatization, which tries to make sense of words whereas stemming removes the common suffixes. But in the end, we decided to go with stemming as per the instructions.

This tokenizing and stemming works pretty well for simple pages that is straightforward. However, there are also a few tokens where it's just one letter, like "s" or "l" which are unexpected. This happens as a side effect of removing special characters like a dash or apostrophe and so on, so it could have performed better if I were to lessen the filtering on tokens.

3 Text Analysis

3.1 Zipf's Law Analysis

For our web crawler, both crawl 1 (Spanish), and crawl 3 (French), seemed to follow the theoretical Zipf's Law pattern pretty well. However, crawl 2 (English) deviates from the predicted Zipf distribution as it starts well below the predicted probability that Zipf's law states. This could mean that the website used for crawl 2 could have some different characteristics that may deviate from Zipf's law. Additionally, all 3 of the crawls had strong deviations from the predicted Zipf distribution towards the right side of each analysis which indicates much lower frequencies of words as the corpora grows.

3.2 Heap's Law Analysis

Similar to Zipf's law, both crawls 1 and 3 closely follow the Heaps Law prediction. In crawl 1 (CervantesVirtual) the empirical values of K = 13.645 and Beta = 0.559 matched the observed values fairly accurately, the only deviations happen towards the ends of the observed values. In crawl 3 (LeFigaro) the empirical values of K = 19.736 and Beta = 0.533 matched the observed values very similar to crawl 1. The main difference between the crawl 1 and crawl 3 is that the observed values for the words in the vocabulary as the words in collection grew fell below the predicted curve whereas in crawl 1 in was above the curve. Crawl 2 (CPP) deviated the most from the predicted curve with values K = 10.809 and Beta = 0.507 with almost no new vocabulary being found after 11000 words in collection. This likely could have happened because the CPP website may have very similar words throughout all of the documents or because of the lower words in collection compared to the other 2 crawls.

4 Work Contributions

Devin Khun: Served as the project manager and developed the tokenizer and stemming feature for each of the web pages.

Daniel Ho: Helped develop the web crawler language detection functionality and using multiple domains

Caden Minniefield: Developed the web crawler feature.

Tony Swank: Developed the code for analyzing stemmed documents and generating Zipf's Law and Heap's Law analysis graphs.

Thet Wai: Responsible for stemming and tokenizing code and documenting the process in the report.

REFERENCES

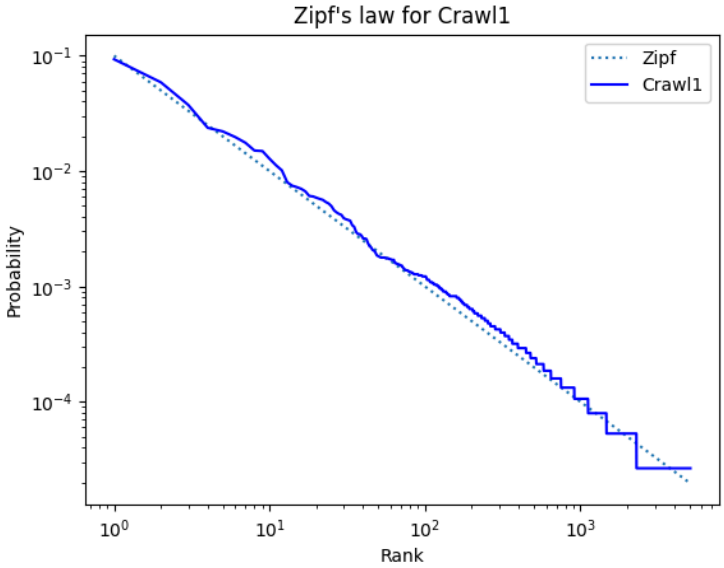
[1] BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

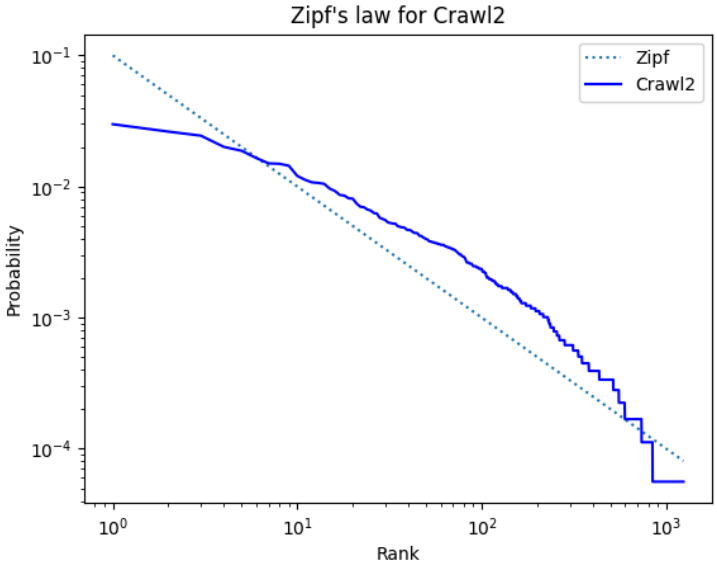
[2] Natural Language Toolkit: https://www.nltk.org/

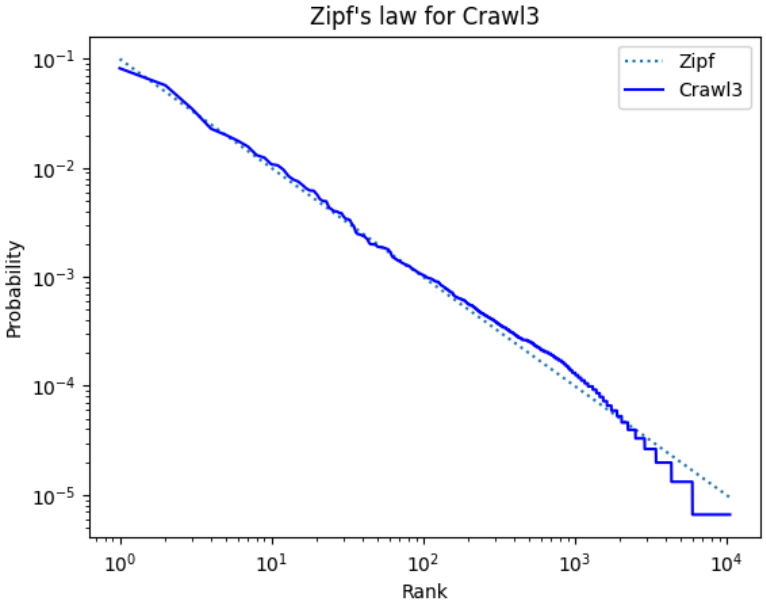
[3] Lingua: https://github.com/pemistahl/lingua-py

A APPENDICES

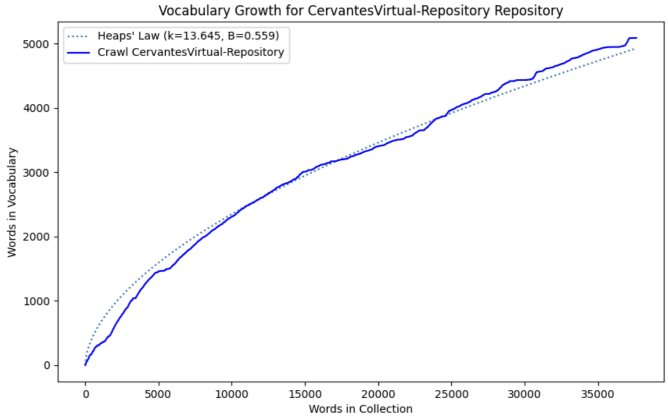
A.1 Zipf's Law Graphs



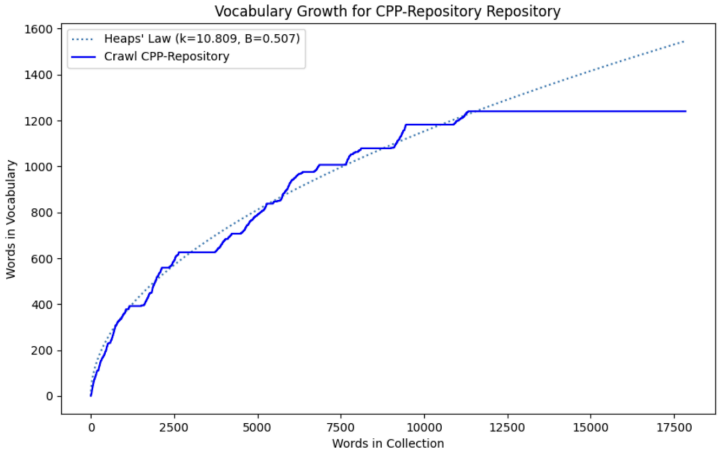



Zipf's law for Crawl3
Zipf
Crawl3
Probability
Rank
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}

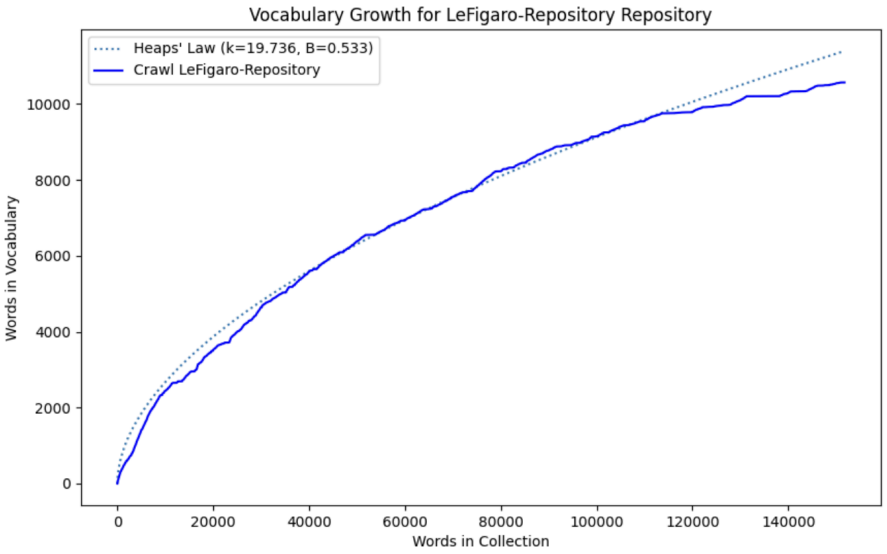
A.2 Heap's Law Graphs



-



-


Vocabulary Growth for LeFigaro-Repository Repository
Heaps' Law (k=19.736, B=0.533)
Crawl LeFigaro-Repository
Words in Vocabulary
Words in Collection
0
2000
4000
6000
8000
10000
0
20000
40000
60000
80000
100000
120000
140000

-