

# **Domain Specific Web Crawler (Python)**

COMP 4601 Final Project  
Prof. Dr. Tony White

Authors:  
Zachary Seguin 101000589  
Azim Baghadiya 101044100

Submission date: April 16th 2020

### **Acknowledgements**

We would like to express our special gratitude and thanks to Dr. Tony White for his teaching and supervision in COMP 4601.

## **Abstract**

This project builds a domain specific web crawler, which is also known as a focused or topic-specific crawler. The goal is to crawl web-pages that are relevant to the domain specified by the user. Crawling performance is usually hindered due to multiple-topic web pages. Additionally, relevant parts of a web page may be ignored due to a low overall relevance score of the webpage. Hence, we explore an improved approach that first classifies the webpage by weighing each term using the ITFIDF approach, then uses a Link Priority Evaluation algorithm to compute the relevance of outgoing links. The algorithm partitions the webpage into content blocks, and the relevance of each content block is determined by computing the similarity score between the unit (content block) vector and the topic vector. The experimental results show that this method is significant and effective for a focused crawler.

## **Introduction**

The gathering, sorting, and parsing of data is becoming more and more important as the growth of network information continues. The internet has become the largest collection of knowledge known to the human race. The sheer size of this internet produces interesting problems to be solved. With this massive stockpile of information, finding relevant information could be like looking for a needle in a haystack. The problem to solve is to collect the relevant information that a user is trying to find, and discard the rest. Crawling web pages allows for the collection of large amounts of information. Web crawlers can either be general-purpose crawlers, or special-purpose crawlers. General-purpose crawlers retrieve a huge amount of information from the entire internet, whereas special-purpose crawlers navigate the internet with a specific goal in mind. A domain specific crawler or focused crawler takes in a domain (some topic) such as baseball, and crawls any page related to the domain meaning that the space in which the crawler operates is restricted by the domain. Having a restricted operational space means that less of the whole internet is crawled which means that, unlike general-purpose crawlers, focused crawlers need a smaller amount of runtime and hardware resources (memory) to complete their task. Due to this, focused crawlers have become a very important tool for gathering information from webpages to be used in applications such as search engines, information extraction, digital libraries, and text classification [4]. The most important part of a focused crawler is text classification as this is the decision making engine that drives the crawler forward. The most common approach for this is TFIDF but this project implements a variation of TFIDF called ITFIDF. This approach separates web pages into sections which have different relevancy. Additionally, links within the sections have their priority evaluated using Link Priority Evaluation (LPE) proposed by Houqing Lu,<sup>1</sup> Donghui Zhan,<sup>1</sup> Lei Zhou,<sup>2</sup> and Dengchao He<sup>3</sup> [4]. In LPE, webpages are partitioned into smaller content blocks which are assessed individually. If a content block is found to be relevant, all of the unvisited links within it are extracted and added to the crawl frontier (url\_queue), otherwise all the links within the content block are discarded.

The remainder of this paper is organized as follows: The Background section explains how focused crawlers work, how we approached implementing one in Python, and discusses some algorithms that can be used to compute important values used in focused crawling. In the

Related Work section, alternative solutions to focused crawling are discussed such as Word2Vec which uses a neural network to construct the necessary weight vectors to determine the relevancy of webpages and discusses other algorithmic approaches to solving the webpage classification problem. The Discussion section provides a critical review of the strengths and weaknesses of our implementation. The Methodology section describes our approach to implementing the focused crawler along with requirements, algorithms used, and architecture. Finally, the Conclusions and Future Work section discusses the performance of this approach (ITFIDF) and future features/changes that could be added to the project to improve the application.

## **Background**

The driving engine behind this project is the web crawler which will visit a set of given URLs and gather data from the pages to be analysed and indexed. When the pages are visited, further URLs are grabbed from elements on the page and added to the “crawl frontier” to be recursively visited. This paper describes an implementation of a topical or focused crawler which means that the crawler will determine the similarity of a web page to a particular query and decide whether to index a page or not depending on the pages’ relevancy to the search domain. The concepts of topical and focused crawling were first introduced by Filippo Menczer and by Soumen Chakrabarti *et al.* [9] which allowed for further development towards search engines as we know them today.

The main decision to be made when implementing a domain specific crawler is between processing speed/efficiency and precision. An example of this would be a focused crawler that can infer the relevancy of a webpage by simply analysing its anchor text, this would allow the crawler to make a decision without needing to download the webpage and scan its contents. The problem with this approach is that anchor text and its context does not provide a lot of information about the topic of the webpage and thus could negatively affect the quality of our search results. The more computationally demanding but more accurate method of determining the relevancy of a webpage to a given domain is to download its html and perform a comparison of its contents to the given domain. This can take a varying amount of time to perform depending on the amount of content on the page, and the tags that will be analysed. For example, looking at headers, titles, and paragraphs on the page will take longer than only looking at paragraphs or other elements on the page like a table or list. For optimal balance between quality of output and processing speed, a mix of the above approaches is recommended.

The main challenge to overcome in a domain specific web crawler is determining the topic of a webpage and its similarity to the reference corpus. Multiple algorithms are available but the core concept of most is using frequency analysis of terms on a vocabulary of terms which are compared to an existing reference corpus of terms. One algorithm to compute similarity between two variables is the Pearson Correlation. According to the Cauchy–Schwarz inequality it has a value between +1 and -1, where 1 is total positive linear correlation, 0 is no linear correlation, and -1 is total negative linear correlation. It is widely used in the sciences [11].

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \quad (\text{Eq.1})$$

where:

- cov is the [covariance](#)
- $\sigma_X$  is the [standard deviation](#) of  $X$
- $\sigma_Y$  is the standard deviation of  $Y$

The above image is the formula for the Pearson Correlation. Another algorithm for information retrieval is TF-IDF (term frequency-inverse document frequency). The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general [12]. However, TF-IDF doesn't consider the effect of the length of different documents on weighting, in order to include such effect, TFC is used. Since TFC is used in our implementation, the formula is discussed in the methodology section. The TFC formula allows for the construction of weight vectors for each unit inside of the document. Once the unit weight vector is constructed, a topic vector must also be constructed (all zeros except for 1 value where domain term appears). Similarity values are computed by comparing the topic vector to the unit vector generated by TFC. Once similarity values are computed, the crawler can decide which urls are most relevant to the topic by sorting by highest Sim(u,v) values.

More recent methods of computing the relevancy of documents utilise neural networks which can perform the task of document classification much more rapidly once a model is trained. One of these tools is Word2Vec which will be discussed in the following section.

## **Related Work**

In the field of information retrieval it is becoming more and more common to solve the issue of document relevance/similarity by using a neural network. The previously mentioned Word2Vec tool creates word embeddings. In word2vec, using the continuous bag of words setting, word embedding vectors are created by a two-layer neural net which tries to guess which word appears in the middle of a context [7]. Using another setting skip-grams, the neural net tries to predict the words that appear around a given word. In either case, initial, random word embeddings are gradually altered by the gradient descent mechanism of neural nets, until a stable set is found [7]. After extensive testing this method seems to produce impressive results with a high level of efficiency. The only issue with using such a method for this particular project is that either a pre-trained model would have to be download alongside the project which could be multiple GBs in size, or a new model would have to be trained using a fairly large reference corpus to ensure that the quality of results is not impacted.

For this reason a different approach was used to compute the topic of a webpage which was modeled after the research paper “An Improved Focused Crawler: Using Web Page Classification and Link Priority Evaluation” by Houqing Lu,<sup>1</sup> Donghui Zhan,<sup>1</sup> Lei Zhou,<sup>2</sup> and Dengchao He<sup>3</sup> [4]. This approach uses the algorithms mentioned in the Background section of this report to generate weight vectors for terms inside of content blocks. The content blocks can

be any grouping of data within the HTML. Once weight vectors are generated, a similarity value is computed by comparing the weight vectors of terms within content blocks to the topic vector. The most relevant content blocks are reviewed and the urls within are extracted and added to the crawl frontier ensuring that only relevant documents are crawled.

## Methodology

The underlying goal is to find the regions of the web page that are relevant to the topic. In this algorithm, we partition the web page into content blocks and represent them as unit vectors. Then we calculate the relevance of each unit vector by computing its similarity with the topic vector. This allows the crawler to draw outgoing links from highly relevant regions from a low overall relevance web page.

To calculate the similarity between links and the topic, we following the following steps:

- the highest priority URL is dequeued from the priority queue (initially, the seed URLs are enqueued with the highest priority)
  - the webpage is downloaded (as HTML) using *urllib* library, and the HTML is parsed using the *beautiful soup* library
  - web page is partitioned into content blocks and the method of similarity measure is used to calculate the relevance of each content block
  - links from the top-k relevant content blocks are drawn and added to the priority queue
  - the similarity of the block is considered to be the priority of the links
- Note* that some links from a relevant block are irrelevant, so a document score is computed for each crawled URL and the ones with a score of 0 are skipped.
- web pages are continuously downloaded for the specified topic/domain until there are no more links in the priority queue or the number of crawled pages reaches a limit

Similarity between a unit vector and the topic is computed using the cosine measure:

$$\text{Sim}(u, v) = \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}| \times |\mathbf{v}|} = \frac{\sum_{t=1}^n w_{t,u} \times w_{t,v}}{\sqrt{\sum_{t=1}^n w_{t,u}^2} \times \sqrt{\sum_{t=1}^n w_{t,v}^2}},$$

$u$  = unit vector,  $v$  = topic vector,  $w_{t,u}$  = weight of  $u$ ,  $w_{t,v}$  = weight of  $v$ .

Weight of each term in the vector uses the TFC weighting equation:

$$w_{t,u} = \frac{f_{t,u} \times \log(N/n_t)}{\sqrt{\sum_{r=1}^M [f_{r,u} \times \log(N/n_r)]^2}}$$

$f_{t,u}$  = frequency of term  $t$  in the unit  $u$ ,  $N$  = # of content blocks,  $M$  = # of terms in the content block, and  $n_t$  # of content blocks where term  $t$  appears at least once.

## Discussion

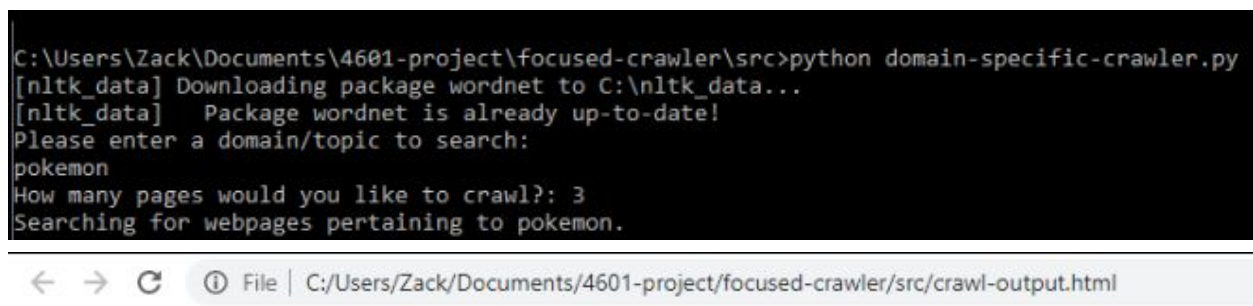
In general purpose crawling, relevant parts of the page may be ignored or anchor text and link-context may be misleading. This may lead to relevant web pages being crawled. Conversely, the domain specific web crawler implemented in this project employs an improved term weighting approach, ITFIDF, which draws on highly relevant web pages to crawl, which turns out to be key strength.

Unlike general purpose web crawlers, that collect enormous amounts of information from a large number of web pages, the domain specific crawler is able to achieve good precision and good recall as the crawl was restricted to a specific domain. Additionally, this approach makes use of a customized web page partitioning algorithm, to pick URLs with highest priority at each step. Here, a URL that is more relevant to the domain of crawl has a higher priority / relevance score, which results in a higher target recall than other crawlers.

Next, let's talk about key weaknesses, which when resolved, can adequately improve the crawler. First, the topic-focused crawler is crawling web pages from a specific domain, that is fed as a seed in the codebase. Currently, the algorithm chooses Wikipedia as the domain for the seed URL. For example, if the algorithm is launched with the following options:

- domain to crawl: pokemon
- # of pages to crawl: 3

the algorithm crawls three webpages that belong to the wikipedia.org domain. Below is the snapshot of the result:



```
C:\Users\Zack\Documents\4601-project\focused-crawler\src>python domain-specific-crawler.py
[nltk_data] Downloading package wordnet to C:\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
Please enter a domain/topic to search:
pokemon
How many pages would you like to crawl?: 3
Searching for webpages pertaining to pokemon.
```

← → ↻ ⓘ File | C:/Users/Zack/Documents/4601-project/focused-crawler/src/crawl-output.html

### Links relevant to pokemon

- <https://en.wikipedia.org/wiki/pokemon>
- [https://en.wikipedia.org/wiki/List\\_of\\_Pok%C3%A9mon](https://en.wikipedia.org/wiki/List_of_Pok%C3%A9mon)
- <https://en.wikipedia.org/wiki/Pikachu>

Figure 1: Launching the algorithm with (pokemon, 3), the result lists the crawled URLs

The problem arises when a wikipedia page links to a lot of its own articles, then the outgoing links that are potentially crawled turn out to be many Wikipedia links. Why does this problem occur? This is because a user input is a domain/topic name, but the crawler must use a seed URL to start crawling. In order to generate a seed URL, we decided to choose wikipedia, because of the abundance of information available through this source. For example, there are

over 6 million English articles alone on wikipedia, so a good start would be to try the following as the seed url:

[www.wikipedia.org/wiki/topic](http://www.wikipedia.org/wiki/topic)

The next drawback is that the program only accepts a single-word domain/topic as input. For instance, the crawler would accept *Baseball* as domain but not *Detective Pikachu*. This ties back to the previous problem, seed URL generation. Although Wikipedia has an abundance of articles, it would be misleading to assume that this format [www.wikipedia.org/wiki/topic](http://www.wikipedia.org/wiki/topic) will always be valid. For instance, [https://en.wikipedia.org/wiki/Detective\\_Pikachu](https://en.wikipedia.org/wiki/Detective_Pikachu) because Detective Pickachu could mean Detective Pickachu (film) or Detective Pickachu (video game). Then, the following would be valid, but not trivial to generate by the program:

- [https://en.wikipedia.org/wiki/Detective\\_Pikachu\\_\(film\)](https://en.wikipedia.org/wiki/Detective_Pikachu_(film))
- [https://en.wikipedia.org/wiki/Detective\\_Pikachu\\_\(video\\_game\)](https://en.wikipedia.org/wiki/Detective_Pikachu_(video_game))

Lastly, another shortcoming is the slow runtime. The crawler seems to run significantly slow, and this could be partly due to a combination of reasons, including the choice of programming language, and the implemented algorithm. Interpreted languages, like Python, are usually slower (ex: compared to Java). However, it's important to note that a key aspect of the algorithm is calculating the similarity between the topic and each content block. In order to calculate the relevance score/similarity of a content block, we must find the weight of each term in the block, which looks at the entire document corpus, and this could increase the runtime and worsen the performance. Running the computationally intensive functions on separate threads or on multiple cores could improve the runtime.

## Evaluation of Performance

This section highlights the experimental results discussed in the referenced paper “An Improved Focused Crawler: Using Web Page Classification and Link Priority Evaluation” by Houqing Lu, Donghui Zhan, Lei Zhou, and Dengchao He.

As part of the experiment, seed URLs and 30 relevant web pages (each) are selected for 10 topics. The key to measuring the performance of the domain specific crawler is to estimate the relevant web pages collection rate. However, the Internet doesn't know what web pages are relevant to a given topic, so measuring true recall may not be possible. Alternatively, we measure harvest rate and target recall values to evaluate the domain specific crawler's performance.

$$\text{Harvest Rate} = \frac{\sum_{i \in V} r_i}{|V|} \quad \text{Target Recall: } \frac{|T \cap C_t|}{|T|}$$

- where V = # of web pages crawled,  $r_i$  = relevance score of web page i with the domain/topic (0/1, where 1 = relevant, 0 otherwise)
- where T = target recall,  $C_t$  = first t web pages crawled



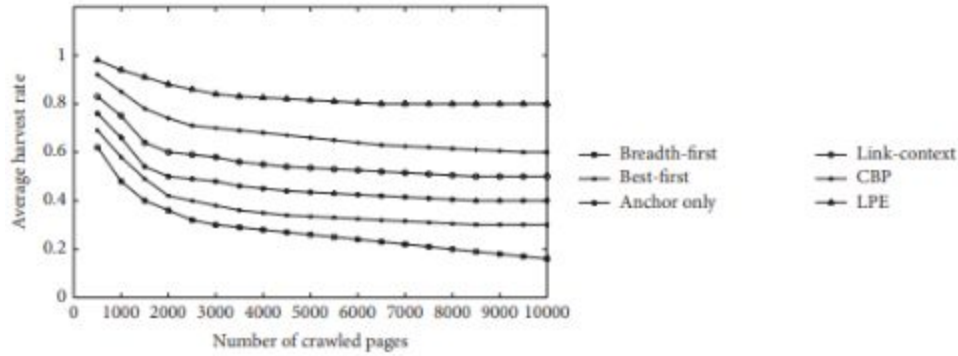


Figure 2: Avg harvest rates for 6 crawlers

The above figure compares the performance of average harvest rate for 6 different crawlers, that crawl 10 topics each. We observe that the LPE crawler crawls more web pages than the other crawlers, and the harvest rate of the LPE crawler is between 0.16-0.8 times higher than its opponents. Hence, it's fair to argue that the LPE algorithm allows our domain specific crawler to crawl more topical web pages than it's comparable competitors.

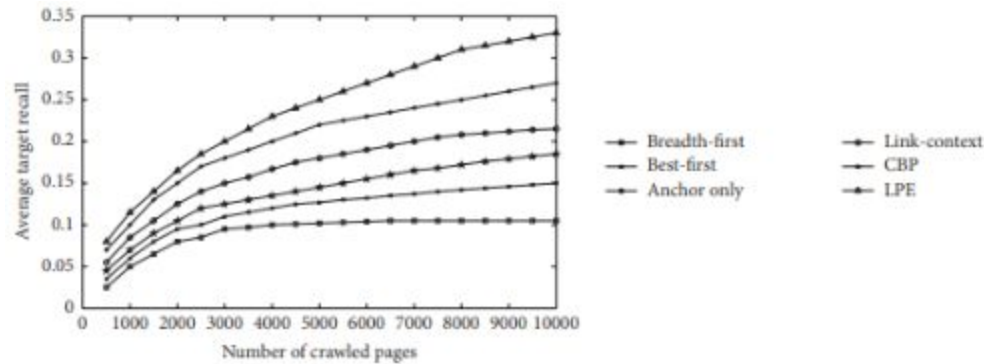


Figure 3: Avg target recalls for 6 crawlers

The above figure compares the performance of average target recall rates for 6 different crawlers, that crawl 10 topics each. We observe that the average target recall rate of the LPE crawler is between 0.16-3.3 times higher than its opponents. Hence, it's fair to argue that the LPE algorithm allows our domain specific crawler to crawl greater qualities of topical web pages than it's comparable competitors.

The experiment results conclude that the LPE crawler crawls better qualities of topical webpages than the other 5 crawlers, and this improves the performance of the focused crawlers.

## **Conclusion and Future Work**

This project implements a domain specific crawler that aims to achieve high collection performance by classifying the webpages and utilizing a link priority evaluation algorithm. An effective strategy is to use ITFIDF as it allows us to consider the different feature distributions based on the page positions when building a web page classifier. Referenced paper, "An Improved Focused Crawler, by Houqing Lu, Donghui Zhan, Lei Zhou, and Dengchao He, evaluates the performance of the classifier that uses ITFIDF against a TFIDF-based classifier with 4 different datasets, and the former outperformed the latter in each case.

If this project were to continue, we could allow the algorithm to use a variety of popular domains for the seed URL(s), and not just Wikipedia. An effective strategy would be to offer the user a number of topics to choose from, instead of asking for a topic as a user input. Crawling on a predefined set of topics would allow us to feed more than one seed URL, by manually picking and feeding URLs of interest pertaining to each topic. Below is a workable example:

| Topic        | Seed URLs                                                                                                                                                                                                                                                                                   |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Basketball   | <a href="https://en.wikipedia.org/wiki/Basketball">https://en.wikipedia.org/wiki/Basketball</a><br><a href="https://www.espn.com/mens-college-basketball/">https://www.espn.com/mens-college-basketball/</a><br><a href="https://www.cbssports.com/nba/">https://www.cbssports.com/nba/</a> |
| Mobile phone | <a href="https://www.carphonewarehouse.com/mobiles">https://www.carphonewarehouse.com/mobiles</a><br><a href="https://pdaplaza.ca/">https://pdaplaza.ca/</a><br><a href="https://en.wikipedia.org/wiki/Mobile_phone">https://en.wikipedia.org/wiki/Mobile_phone</a>                         |

Table 1: Examples of seed URLs for future work

## Appendix:

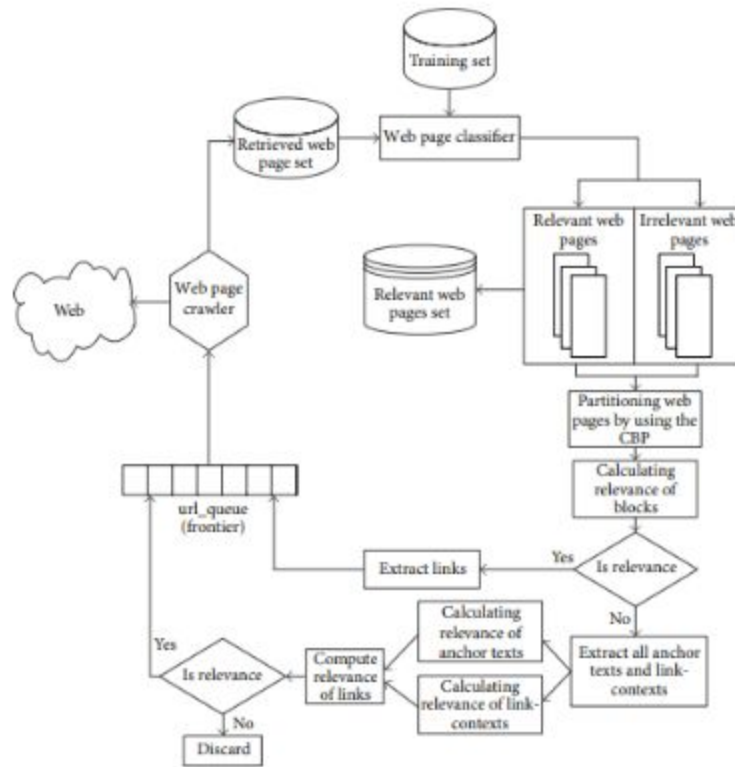


Figure 4: Architecture of the domain specific crawler

## **References:**

- [1]. Jabeen, Hafsa. "Stemming and Lemmatization in Python." *DataCamp Community*, 2018, [www.datacamp.com/community/tutorials/stemming-lemmatization-python](http://www.datacamp.com/community/tutorials/stemming-lemmatization-python).
- [2]. BeautifulSoup Documentation¶. (n.d.). Retrieved from <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- [3]. JosJos 4511 silver badge55 bronze badges, ARMANARMAN 3, & PushkrPushkr 2. (1966, December 1). BeautifulSoup - getting tag contents. Retrieved from <https://stackoverflow.com/questions/42328593/beautiful-soup-getting-tag-content>
- [4]. Lu, Houqing, et al. "An Improved Focused Crawler: Using Web Page Classification and Link Priority Evaluation." *Mathematical Problems in Engineering*, Hindawi, 19 May 2016, [www.hindawi.com/journals/mpe/2016/6406901/](http://www.hindawi.com/journals/mpe/2016/6406901/)
- [5]. Google Code Archive - Long-term storage for Google Code Project Hosting. (n.d.). Retrieved from <https://code.google.com/archive/p/word2vec/>
- [6]. RDPDRDPD 14133 silver badges77 bronze badges, galoosh33galoosh33 1, & user79309user79309. (1967, January 1). Using LDA to calculate similarity. Retrieved from <https://stats.stackexchange.com/questions/271359/using-lda-to-calculate-similarity>
- [7]. Grefenstette, G. G., & Muchemi, L. (2016, May 31). Determining the Characteristic Vocabulary for a Specialized Dictionary using Word2vec and a Directed Crawler. Retrieved from <https://arxiv.org/abs/1605.09564>
- [9]. Jphcoi. (n.d.). jphcoi/crawtext. Retrieved from <https://github.com/jphcoi/crawtext>
- [10]. Web crawler. (2020, March 25). Retrieved from [https://en.wikipedia.org/wiki/Web\\_crawler](https://en.wikipedia.org/wiki/Web_crawler)
- [11]. Naive Bayes classifier. (2020, April 12). Retrieved from [https://en.wikipedia.org/wiki/Naive\\_Bayes\\_classifier](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)
- [12]. Naive Bayes classifier. (2020, April 12). Retrieved from [https://en.wikipedia.org/wiki/Naive\\_Bayes\\_classifier](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)
- [13]. Pearson correlation coefficient. (2020, April 15). Retrieved from [https://en.wikipedia.org/wiki/Pearson\\_correlation\\_coefficient](https://en.wikipedia.org/wiki/Pearson_correlation_coefficient)
- [14]. Tf-idf. (2020, April 15). Retrieved from <https://en.wikipedia.org/wiki/Tf-idf>