Building Large Corpora from the Web Using a New Efficient Tool Chain

Roland Schäfer, Felix Bildhauer

German Grammar, SFB 632/A6
Freie Universität Berlin
Habelschwerdter Allee 45, 14195 Berlin
roland.schaefer@fu-berlin.de, felix.bildhauer@fu-berlin.de

Abstract

Over the last decade, methods of web corpus construction and the evaluation of web corpora have been actively researched. Prominently, the WaCky initiative has provided both theoretical results and a set of web corpora for selected European languages. We present a software toolkit for web corpus construction and a set of siginificantly larger corpora (up to over 9 billion tokens) built using this software. First, we discuss how the data should be collected to ensure that it is not biased towards certain hosts. Then, we describe our software toolkit which performs basic cleanups as well as boilerplate removal, simple connected text detection as well as shingling to remove duplicates from the corpora. We finally report evaluation results of the corpora built so far, for example w. r. t. the amount of duplication contained and the text type/genre distribution. Where applicable, we compare our corpora to the WaCky corpora, since it is inappropriate, in our view, to compare web corpora to traditional or balanced corpora. While we use some methods applied by the WaCky initiative, we can show that we have introduced incremental improvements.

Keywords: web corpora, corpus evaluation, web crawling

Corpus	N Documents	N Tokens
SECOW2011	1.91	1,750
ESCOW2012	1.30	1,587
DECOW2012	7.63	9,108

Table 1: Document and token counts (million) for selected COW corpora

1. Introduction

In this paper, we describe a new tool chain for the processing of web data for the purpose of web corpus construction. Furthermore, a growing set of very large web corpora, which we constructed using the aforementioned tools, is introduced and evaluated (Table 1 lists a selection of these corpora and their respective sizes). We proceed by discussing the methods of data collection in 2., describing the methods implemented in the software in 3. before reporting results of the evaluation of the final corpora and the quality of the software in 4.

2. Data Collection

The purpose of the project described here is the construction of very huge comparable web corpora in a large number of languages. Although we share with Kilgarriff and Grefenstette (2003, p. 340) a more or less agnostic position towards the question of representativeness of web corpora, we still consider it crucial to define the population of which a corpus is intended to be a sample. We sample from all WWW documents under a given national TLD, written in an official language of the respective country and containing predominantly connected text. From each document, we attempt to extract all paragraphs of connected text, finally making sure that the majority of the text within the document occurs only once in the final corpus (cf. 3.4.). Since the given population is unknown to a large extent, proportional sampling is out of the question, and large random samples are the only valid option. How a random sample from the web can be approximated is not a settled question. In web corpus construction, it is customary to send large numbers of requests to search engines using tuples of mid-frequency word forms as search strings (Baroni and Bernardini, 2004). Either only the URLs returned are then harvested (BootCaT method), or they are used as seed URLs for a crawler system. We apply the second method, since corpora of roughly 10^{10} tokens cannot practically be constructed from search engine queries alone. Also, the BootCaT method, as opposed to an approach using specialized crawler software, does not allow to effectively control the politeness of the harvesting process (Manning et al., 2009, 444ff.). As a crawler we use Heritrix 1.4 configured roughly as described by Emerson and O'Neil (2006). Seed URLs were taken from Yahoo until their discontinuation of the free API access, after which we switched to Microsoft Bing. Besides the fact that the Yahoo API discontinuation proves that total dependence on search

engines is generally an unsafe strategy, we would like to point out some problems with search engine results and argue for much stronger reliance on very large web crawls for linguistic purposes.

The randomness of web corpora was examined successfully in the BootCaT/WaCky community from a linguistic viewpoint. For example, Ciamarita and Baroni (2006) evaluate randomness of corpora based on linguistic features. The most basic measure of randomness of both seed URLs retrieved from a search engine and documents downloaded by a crawler, however, is its bias towards certain hosts. Even though a web corpus (or a set of search engine results) might never be called balanced, it should count as biased if it proportionally contains a huge number of documents from only a few hosts. It is known from papers such as Bar-Yossef and Gurevich (2006) that obtaining random samples from search engines is not a simple matter, especially when there is only one major search engine left which allows free API access for large numbers of queries. To our knowledge, nobody has ever constructed web corpora making efforts such as described by Bar-Yossef and Gurevich (2006). Since the method described in that paper only approximates random samples from search engine indices and not from the web itself, even following the procedure would not ensure random samples in the desired sense. The customary method, however, does not even ensure bias-free sampling from a search engine's index.

Our experiments so far have shown that a large number of hosts which never occur in search engine results can be discovered through long-term crawling. In Figures 1 and 2, experiments for the `de` and `se` domain are evaluated with respect to their host bias. The graphs cumulatively plot for a random sample of 10,000 URLs from either a set of unique seeds gathered from the search engines (Yahoo for `se`, Bing for `de`) and the document URLs of a final corpus derived from these seeds (which has undergone the cleaning procedures explained in 3.) how large a proportion r of the total URLs comes from to the n most popular hosts in the whole sample. The seed URLs were queried using 3-tuples of content word forms (nouns, verbs, adjectives, adverbs) ranked 1,001 through 6,000 in frequency lists either from existing corpora or from previous crawl experiments. A maximum of 10 results per tuple were requested from the search engine. The crawler ran for approximately seven days for the `se` domain, and for 28 days in the `de` case. The total sizes of the seed sets N_{seeds} and the total number of document URLs in the corpus N_{corpus} is given for each experiment. The third curve in each graph plots the proportion of documents in the final corpus which stem from the top n hosts from the seed set. It is obvious that the experiments were successful to different degrees.

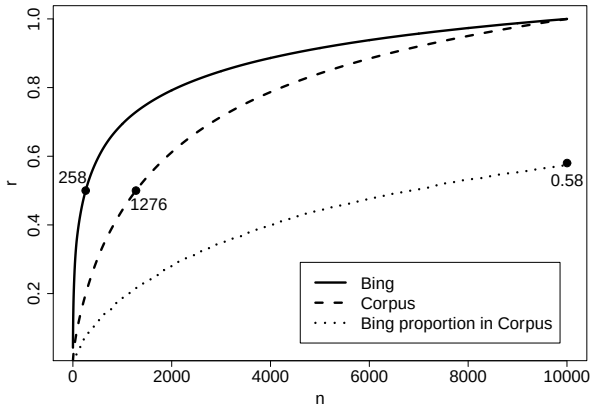

Figure 1: Host bias for DECOW2012 with Bing seeds, $N_{corpus} = 7,759,892$, $N_{seeds} = 912,243$

Using an extremely large set of unique seed URLs (over 900,000) causes the seed set to be comparatively less biased: in the DECOW2012 seeds (Figure 1), 50% of the URLs come from the 258 most popular hosts, while in the SECOW2011 (Figure 2) seed set, figures is much lower: 29 hosts contribute 50% of the seed URLs.[1] A longer crawling time helped to bring down the host bias of DECOW2012 significantly, such that in the final corpus, 50% of the documents in the sample come from 1,276 hosts. Also, the 10,000 most prominent hosts in the DECOW2012 seed set contribute only 58% of the documents from the top 10,000 hosts in the final corpus.

The graph for SECOW2011 (Figure 2) in fact shows a completely unsuccessful crawl experiment, where an already heavily biased seed set resulted in even more biased crawl results. The seeds as returned by Yahoo were extremely biased to begin with, and the crawl, using a breadth-first strategy, got stuck trying to exhaustively harvest the few hosts strongly represented in the seeds. This resulted in an interesting corpus, but definitely not in a general-purpose corpus. It contains 1,443,452 documents (75.5% of the total documents) from the host `blogg.se`. Since in this case, the host is strongly tied to a certain type/genre of text and communication (blog posts), it is obvious that host bias can be directly linked to linguistic biases.

To illustrate further that the seed set and the crawling process both play a role, we now compare the re-

[1] Due to the shutdown of Yahoo's API access, it is now impossible to find out whether this is also due to different biases between Yahoo and Bing.

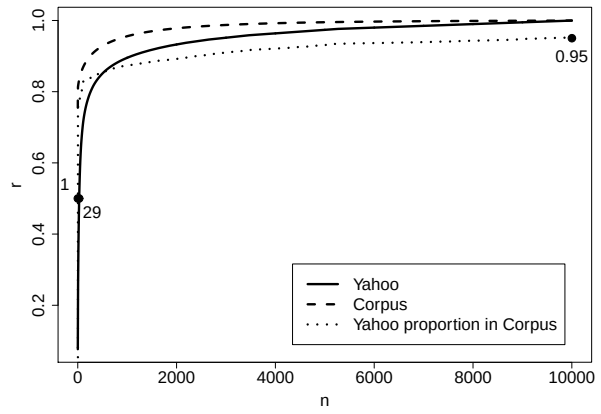

Figure 2: Host bias for SECOW2011 with Yahoo seeds, $N_{corpus} = 1,912,757$, $N_{seeds} = 204,872$

sults to a version of the deWaC corpus from which we removed near-duplicates using our shingling tools (as described in 4.2.) in Figure 3. Only 8,631 hosts are considered, because the seed set contains only this many hosts. The deWaC crawl was based on a seed set where each host was represented only once, resulting in a linear host bias graph. During the 10 day crawl using a breadth-first strategy, however, the crawler harvested mainly the hosts from the seed set, leading to a much stronger bias. This corroborates our view that both the seed sets and the crawling strategies have to be chosen with greater care than is customary.

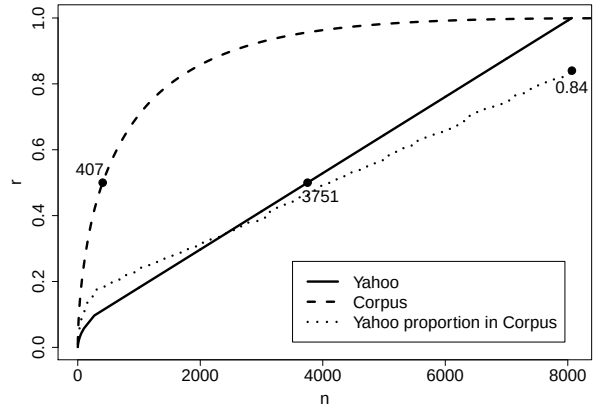

Figure 3: Host bias for shingled deWaC with Yahoo seeds, $N_{corpus} = 1,501,076$, $N_{seeds} = 8,631$

Finally, host bias is reflected also in the average number of documents per host, which is 20 for DECOW2012, 30 for ESCOW2012 and 158 for the shingled deWaC. Since we can safely assume that the population of hosts in the respective TLDs is much larger than the number of hosts represented the corpora, we consider strong host biases unacceptable.

In our view, these results allow at least the following preliminary conclusions:

1. Relying solely on search engine queries might introduce severe bias towards certain hosts and thus to certain types of document or register. Relying on search engines is also unwise because it is foreseeable that no search engine provider will allow free API access in the long run.
2. Long, extensive web crawls can help to alleviate a host bias in the seeds.
3. Crawler software should be adapted or rewritten to offer crawling strategies optimized for better randomness of corpus crawls. Simply implementing a per-host quota in the crawler (a limit to n documents per host) is not a wise idea, since the limit should be imposed on n good documents coming from a host, but the quality of the documents can currently only be assessed after post-processing.

Since specific crawler software for corpus crawls has not been written, and our own crawler is still in planning stage, we suggest to provisionally use existing crawler software and doing very long crawls to create corpora like DECOW2012.

3. Post-Processing

In this section, we describe the function of our own tool chain, which fills the gap between the crawler and linguistic post-processing and indexing tools. The input is typically a file (or directory containing multiple files) which contain(s) HTML/XML documents. The tools are preconfigured for ARC files as created by the Heritrix 1.4 crawler, but they can be reconfigured to deal with alternative file formats and to apply external pre- and post-processing commands to the input files. They are written in a modern variant of ObjectPascal[2], which also allows for parallelization of the more demanding algorithms.

3.1. Basic Cleanup

Under basic cleanup, we subsume the trivial tasks of HTML and script removal, codepage conversion, etc. as performed by the *texrex* tool. HTML input is read from a stream, parsed on the fly, cleansed from markup and scripts, and buffered for further processing. Since the tool chain is optimized for ISO-8859

[2] `http://www.freepascal.org/`

languages, a simple conversion from UTF-8 multibyte characters to corresponding ISO-8859 characters as well as a complete HTML entity conversion is also performed on the fly. The software deals with faulty markup by favoring cleanliness of output over conservation of text and discards the whole document in case of serious unrecoverable errors.

Based on a configurable list of markup tags, a simple paragraph detection is performed, inserting paragraph breaks wherever one of the configured tags occurs. Optionally, the results of basic cleansing are then subjected to a second pass of markup removal, since many web pages contain literal markup pieced together with entities (`<br>` as `<br>` etc.). Perfectly identical subsequent lines are also removed, and excessively repeated punctuation is compressed to reasonable sequences of punctuation tokens.

3.2. Boilerplate Removal

The removal of navigational elements, date strings, copyright notices, etc. is usually referred to as boilerplate removal. The WaCky method (Baroni et al., 2009), simply speaking, selects the window of text from the whole document for which the ratio of text-encoding vs. markup-encoding characters is maximal. This has the advantage of selecting a coherent block of text from the web page, but also has the disadvantage of allowing intervening boilerplate to end up in the corpus. Also, many web pages from blog and forum sites contain several blocks of text with intervening boilerplate, of which many can be lost if this method is applied.

Therefore, we decided to determine for each paragraph individually whether it is boilerplate or not. The decision is made by a multi-layer perceptron as implemented in the FANN library (Nissen, 2005). The input to the network is currently an array of nine values which are calculated per paragraph:

1. the ratio of text encoding characters vs. markup encoding characters,
2. the same ratio for a window extended by one paragraph in both directions,
3. the same ratio for a window extended by two paragraphs in both directions,
4. the raw number of text (non-markup) characters,
5. the ratio of uppercase vs. lowercase characters,
6. the ratio of non-letters vs. letters in the text,
7. the same ratio for a window extended by one paragraph in both directions,
8. the same ratio for a window extended by two paragraphs in both directions,
9. the percentile of the paragraph within the text mass of the whole document.

It is obvious that value 5 is language-dependent to some extent. However, after training, the feature turned out to be weighted so lightly that the network performed equally well for English, German, French, Spanish, and Swedish and we decided to use one network for all languages. This pre-packaged network was trained on decisions for 1,000 German paragraphs coded by humans using the graphical tool *textrain* included in the package. As the only coding guideline which turned out to be practical, we defined:

1. Any paragraph containing coherent full sentences is text,
2. any heading for such text (i. e., not page headers etc.) is text,
3. everything else is boilerplate.

Networks can be trained with user-coded data using the included *texnet* tool. The network must use a symmetric Gaussian output activation function, which produces real numbers between 0 and 1. The user can then evaluate the results and determine a cutoff point in $[0..1]$ below which a paragraph will actually be considered boilerplate in a production run.[3] The measures of accuracy for the pre-packaged network depending on the selected cutoff are shown in Figure 4, which measures the results of an application of the pre-packaged network to 1,000 unseen German paragraphs coded using identical guidelines. The overall accuracy is quite adequate compared to the approaches described in (Baroni et al., 2008). For the COW corpora, we selected the cutoff with the highest F-score (harmonic mean of precision and recall), which is 0.48.

3.3. Detecting Connected Text

Language identification on the document level has been around since the 1960s, and detecting a known language is a simple text-book matter. Since our corpora are designed mainly for research in theoretical linguistics, we were not satisfied with simple language detection, for example based on character n-grams. Many documents on the web contain some text in a specific language, but it is often not connected text but tables, tag clouds, or any other non-sentence material. For example, we ran lists of German content words without function words (artificial tag clouds) through standard language identifiers such as the one included

[3] The *texrex* tool provides functions to create evaluation output where decisions are logged but not executed.

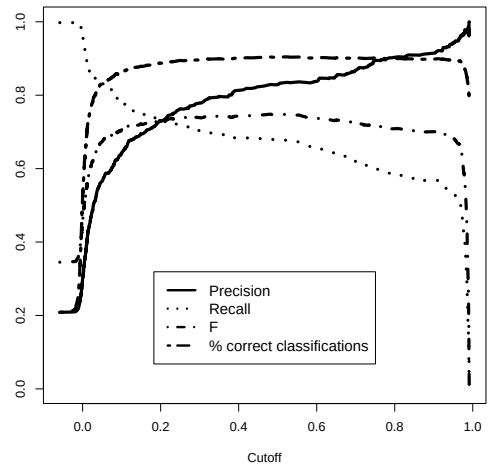

Figure 4: Quality of boilerplate removal depending on cutoff value (pre-packaged network on 1,000 unseen German paragraphs)

in the LingPipe software.[4] For any such list, it recognized German with $P(Category|Input) = 1$. This is clearly undesirable for the intended kind of corpus.

The WaCky corpora were cleaned by allowing only documents which contain ten types and 30 tokens of function words from the target language (Baroni et al., 2009). This is simple and effective in detecting connected text, but it makes the result depend heavily on the length of the input documents and fails to remove long documents with a mix of languages of which the target language represents a large but still not the largest portion. We use the same approach, but without the dependence on absolute token frequencies of certain types. A separate tool (*texprof*) is available which generates a profile for a given language. It takes a specifiable number m of training documents and applies a minimal tokenizer to them (anything in between two non-letters in ISO-8859 is a token). For a specifiable number n of the most token-frequent types $t_{1,..,n}$ across all training documents, it calculates the weighted arithmetic mean of the relative frequencies of these types and the weighted standard deviation (weighted by document length). If $\mu(t)$ is the weighted mean for type t in the training documents, $\sigma^2(t)$ the corresponding weighted standard deviation, and $f(t, d)$ the relative frequency of t in d, then we can (in a production run) for each document d calculate a standardized summed negative deviation $B(d)$ (from the mean in the training documents) for all t_n.

$$z(t, d) = \frac{\mu(t) - f(t, d)}{\sigma^2(t)} \quad (1)$$

$$b(t, d) = \begin{cases} z(t, d) & \text{if } z(t, d) > 0 \\ 0 & \text{else} \end{cases} \quad (2)$$

$$B(d) = \sum_{i=1}^{n} b(t_i, d) \quad (3)$$

Finally, all documents are removed for which $B(d)$ exceeds a certain threshold. As training documents for the COW corpora, we used a hand-selected subset of the downloaded documents with $m \approx 400$, $n = 10$. We discarded documents with $B(d) > 10$.

3.4. Perfect and Near Duplicate Removal

Duplicate removal is done in two steps. The *texrex* tool performs perfect duplicate removal by creating for each document an array of 128 characters, which are evenly distributed over the whole document. These arrays are stored using a fast string hash table in memory, and documents corresponding to known arrays are discarded.

To perform near duplicate removal, there is a separate tool chain providing a native implementation of w-shingling (Broder et al., 1997) without clustering. Documents are tokenized by the parallelized *teshi* tool, which then forms the set of token-n-grams (or w-shingles) for each document. The shingles are hashed with m 64-bit Rabin hash functions (Rabin, 1981), each seeded with a 64-bit representation of a different irreducible polynomial to approximate the random permutations of the hashes required according to Broder et al. (1997). The document's w-shingling is the set of minimal hashes for each m. The *tender* tool then sorts the shingle database and calculates the number of shingles shared between each document. From document pairs with a number of shared shingles higher than a controlable threshold $t \cdot m$, the shorter one is finally removed by a separate tool. For the COW corpora: $n = 5$, $m = 100$, $t = 0.05$.

3.5. Software Performance

The performance of the software is major a aspect in the construction of very large corpora. For DECOW2012, *texrex* (cf. sections 3.1. to 3.3. plus perfect duplicate removal), which is not parallelized so far, ran for 8.8 CPU days on a Xeon 5160 at 3.00 GHz, processing 130,602,410 input documents (170 input documents per second). The process, however, was slowed down due to compression/recompression of the data with *gzip*.

[4] See `http://alias-i.com/lingpipe/`. The method implemented is based on (Tehan, 2000) according to the manual.