Common Crawled web corpora

Constructing corpora from large amounts of web data

Kjetil Bugge Kristoffersen

Thesis submitted for the degree of
Master in Informatics: Programming and Networks
(Language Technology group)
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2017

Common Crawled web corpora

Constructing corpora from large amounts of web data

Kjetil Bugge Kristoffersen

Abstract

Efforts to use web data as corpora seek to provide solutions to problems traditional corpora suffer from, by taking advantage of the web's huge size and diverse type of content. This thesis will discuss the several sub-tasks that make up the web corpus construction process – like HTML markup removal, language identification, boilerplate removal, duplication detection, etc. Additionally, by using data provided by the Common Crawl Foundation, I develop a new very large English corpus with more than 135 billion tokens. Finally, I evaluate the corpus by training word embeddings and show that the trained model largely outperforms models trained on other corpora in a word analogy and word similarity task.

Acknowledgements

I would like to thank my supervisors Stephan Oepen and Erik Velldal. From the feedback and editing, to taking hours from a busy afternoon to assist me when I was stuck – this thesis would not have been without their guidance.

Thanks to Helene Holter and Henrik Hillestad Løvold, who have made the last year a year of more breaks and coffee than I originally thought was either clever or healthy, respectively. Every day was a bit brighter because of you.

Additionally, I would like to extend my gratitude to the developers and creators of the tools and datasets that were used in this thesis, and for making them available open source.

To the largest sponsor of this work and the work leading up to it, without whom my education would not be possible, a large thank you to the Norwegian welfare state and its policy of free education for all.

Finally, to my girlfriend Ane Granli Kallset: Thank you for your support and understanding, for your love and cups of tea, and for urging me to join late nights out and birthday parties on days when I thought there was no time for such. I love you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A corpus is a collection of texts, and is used for a multitude of tasks in Natural Language Processing, and also by other users such as lexicographers. Corpora are often used statistically to be able to say something about the corpus, or even the language, by for instance using them for creating language models or by providing them as data for unsupervised machine learning tasks to represent words as low-dimensional vectors. The construction of corpora that strike a balance of size and 'quality' (see below) is a methodological and engineering challenge that has received much attention for the past several decades and still today.

The World Wide Web contains vast amounts of text, and therefore offers the possibility to create large and possibly high-quality text collections. The data retrieved from the web are different types of documents – often HTML documents – that need to be refined or cleaned before they can be used. There are several sub-tasks with varying degrees of complexity and scalability that are part of this data refinement, like for instance removing markup and extracting text, removing duplication, and removing so-called *boilerplate* (see below).

The latter task is an example of the complexity that is in large part idiosyncratic to the construction of corpora based on web data. In addition to a web page's main text content, there will often exist elements that are to varying degrees difficult to assess the relevancy of. Elements like navigational menus, timestamps, advertisements, button labels, etc. will intuitively introduce more noise (than linguistically relevant content) if added to a corpus. Too much of this noisy data could affect the models created from such a corpus negatively. Identifying what is relevant content and what is not is challenging enough for humans, and even more difficult for computers.

Another challenging aspect of constructing large corpora from the web is the scale of the data. For example, my project takes as its point of departure snapshots of web content distributed by the Common Crawl Foundation (see below), and just reading through all files of one such snapshot sequentially, without any actual processing of the data, takes about 18 hours, which is not a time-consuming task relative to the ones required to refine the data. Therefore, to construct large corpora on such a scale, parallel computing is required.

The Common Crawl Foundation is a non-profit organisation that provide monthly crawls of the web for free. This thesis use one of these monthly crawls to construct a very large web corpus – enC^3, or the **En**glish **C**ommon **C**rawl **C**orpus – consisting of over 135 billion tokens. To my knowledge, the constructed corpus is the largest web corpus described or available, with the next largest, enClueWeb (Pomikálek, Jakubícek, and Rychlý, 2012), consisting of 82 billion tokens. The corpus is then evaluated by training low-dimensional word vectors, evaluating those, and comparing the results to word vectors trained on other corpora.

The results of the thesis are concrete contributions to the NLP community: A large corpus constructed from web data, as well as the word vectors trained on the corpus using the GloVe model (Pennington, Socher, and Manning, 2014). Both of these resources are made available through the community repository launched by Fares, Kutuzov, Oepen, and Velldal (2017). Additionally, I provide an in-depth assessment of the Common Crawl, including a critical assessment of its text-only versions. Also, the corpus construction process is documented thoroughly, to encourage and lower the bar for replication. Finally, the tools made to analyse and process the data are made publicly available[1].

The structure of the thesis is as follows:

In **chapter 2**, desiderata for corpus construction are reviewed from various angles and 'traditional' approaches are contrasted with web-based ones. Additionally, the development of the web corpus construction task and its subtasks is described. Finally, the chapter presents an overview of previous work in the field, and how web corpora have been evaluated.

Chapter 3 provides a systematic review of what the Common Crawl contains, and especially of the extracted text-only files that are already provided by the Common Crawl. It further compares a selection of these extracted text files to the corresponding raw web data, refined by the process described in chapter 4.

Chapter 4 discusses the requirements for the process of corpus construction, and I review a selection of candidate tools and pick one. The chosen pipeline is described, together with the process of configuring and scaling up the tools to be able to process large quantities of data in an acceptable amount of time. Additionally, the post-processing steps of extracting raw non-boilerplate text from the final corpus and tokenising and sentence segmenting said text is discussed. Finally, the constructed corpus is presented in its three representations: XML, text and a popular, de-facto standard corpus format for NLP – CoNLL.

Chapter 5 discusses how the constructed corpus can be evaluated, and evaluates the corpus by training dense low-dimensional word vectors – so-called word

[1]https://github.com/kjetilbk/enc3tools

embeddings – using the corpus as input. These word embeddings are then evaluated using a word analogy task and word similarity tasks. The results are compared to the results of word vectors trained on two other corpora, one based on Wikipedia and news text and the other on web data. Finally, the corpus constructed in this thesis is shown to outperform the two others in the analogy task, and in a majority of the word similarity tasks.

Chapter 6 concludes and sums up the thesis, and discusses future work related to the thesis.

Chapter 2

Background

To understand the motivation behind the project, one has to understand why and how text corpora are used, especially within the fields of natural language processing (NLP) and corpus linguistics.

2.1 Text corpora

There are differences in opinion as to what constitutes a corpus. Kilgarriff and Grefenstette (2003) describe two of these opinions. One is presented by McEnery and Wilson (2001), and states that while a corpus in principle can be as simple as "any body of text", they believe there to be more inherent meaning to the word than that, and that qualities such as representativeness and machine-readability play a part in whether the collection of texts is a corpus or not.

Kilgarriff and Grefenstette (2003, p. 2) state:

> McEnery and Wilson (2001) mix the question "what is a corpus?" with "what is a good corpus (for certain kinds of linguistic study)", muddying the simple question "is corpus x good for task y?" with the semantic question, "is x a corpus at all?"

While I will discuss representativeness and other qualities of corpora when examining how one evaluates constructed corpora in section 2.4, I will, like Kilgarriff and Grefenstette (2003) simply consider a corpus "a collection of texts".

2.1.1 Usage of corpora

Generally, corpora are often analysed using computers to obtain knowledge about language. Specifically, one often uses statistical measures counted from the corpus, like for instance how often tokens (words) occur and which tokens they occur next to. From these measures, you can for instance create *language models*, where you can tell the probability of a phrase based on the corpus, you can do *statistical machine translation* (where computers read corpora with the same texts, but in different languages (parallel corpora)) to learn how to translate texts automatically, etc.

In the past few years, neural networks have become increasingly popular as machine-learning models to solve a multitude of NLP tasks (Goldberg, 2015; Collobert et al., 2011). These networks often require dense vectors as input. Hence, the use of corpora as unlabeled data in unsupervised learning to represent words as dense vectors has been increasing (Collobert and Weston, 2008; Mikolov, Chen, Corrado, and Dean, 2013; Pennington et al., 2014, and section 5.2).

Other users of corpora include lexicographers, the people who write dictionaries. To be able to provide explanations for what words mean and how they work together to form sentences, lexicographers use corpora to extract said information. This can be so-called concordances – extracts of a specific word being used in a corpus, together with its immediate context – or collocations, words that have a tendency to go together based on the corpus.

Often, the goal of these tasks is to be able to say something about the corpus, or even try to say something about the language in general. This is a challenge, as corpora are collections of real-world text – a subset of the language. Consequently, they do not contain every single word or word sequence of a language, nor do they have the same relative token frequency as the language as a whole. Because of this, rare word sequences in the language might not show up even once in the corpus. A word sequence that does not show up once in the corpus is impossible to separate from word sequences that are not actual words in the language.

2.1.2 Towards larger corpora

Due to the inherent productivity of language, it is impossible to see every ‘language event’, no matter the size of the corpus. However, a larger amount of tokens would increase the likelihood of rarer words showing up in the corpus, thus increasing the amount of non-zero counts, and counteracting the problem of *data sparsity*. Furthermore, a larger amount of tokens means having a larger sample of the population, which will possibly bring more precise statistics. Banko and Brill (2001) explore the performance of a number of machine learning algorithms when increasing the data size, and make the argument that getting more data for the machine-learning algorithms will make more of a difference than fine-tuning the algorithms.

While having more data might not bring better results in all cases, it is a clear motivation for having larger corpora. Unsupervised machine learning tasks that does not require labeled data will especially be able to benefit directly from increased corpus sizes.

Over the years, manually constructed corpora have been increasing in size. From the one million word Brown corpus in the 1960s, the COBUILD project had eight million words in the 1980s, and in 1995 the British National Corpus was released with 100 million words.

A seemingly obvious candidate for a very large collection of textual content is the world wide web, due to its vast size.

2.2 Web corpora

2.2.1 The beginning of the web as corpus

Because the web has so much data, and because so much of it is textual, there have been increasing efforts to use web data as a corpus. Early efforts retrieved both token and phrase frequencies from querying commercial search engines, while others gathered and grouped a lot of 'relevant' documents. Early works of using search engine hit counts are, for example, the usage for word sense disambiguation (Mihalcea and Moldovan, 1999), or (probably most cited) for obtaining frequencies of bigrams that were unseen in a given corpus (Keller and Lapata, 2003). Using one document as a search query, Jones and Ghani (2000) grouped the resulting documents together to create a corpus in the input document's language. Parallel corpora have been created by grouping documents on the web that were equal, but in different languages – like the same manual translated to both the target languages, or a web site offering two identical versions but with different languages depending on the top-level domain (Resnik, 1999).

2.2.2 Hit counts are not a traditional corpus

While using the web in this way to solve NLP problems is a way of using the web as a corpus, these uses are different from using a traditional corpus. Hit counts for words and phrases from commercial search engines, like mentioned above, have been shown to give desired results for some tasks, but this technique is a lot more difficult and time-consuming to use than a regular corpus (Bernardini, Baroni, and Evert, 2006). Additionally, search engine hit counts do not give you as much information as using a traditional corpus does, as the search engines do not part-of-speech-tag or give you the context of the hits. Also, the frequencies you receive are (approximations of) the number of documents the words are in, and not the number of word occurences (Kilgarriff, 2007). The Google N-gram corpus makes searchable N-grams (albeit only up to n = 5) frequencies available, and began providing part-of-speech tags from January 2016. However, the corpus still has the same drawbacks with respect to lack of context that other hit count based methods suffer from.

2.2.3 Constructing large corpora from the web

Work in the field of constructing very large collections of texts from the web, formatted and used like traditional corpora, cannot be attributed to one group or person, but was described as early as by Kilgarriff and Grefenstette (2003). The early descriptions were concretised when Baroni and Bernardini (2004) created the BootCaT toolkit, a software collection constructing a specialised web corpus based on user-input search terms.

WaCky (*Web as Corpus kool ynitiative*) continued this work in 2005, and tried to identify and discuss the subtasks that went into the task of web corpus construction. This coincided with the beginning of the yearly Web as Corpus workshops, organised by The Special Interest Group on Web as Corpus (SIGWAC) of the Association for Computational Linguistics (ACL). These workshops, among a few others, have since been progressing the research on constructing a traditional-like corpus from the web.

Below I will give a summary of some efforts and tools that have tried to solve this task. The detailed differences in their approaches to the sub-tasks will be discussed in section 2.3.

BootCaT

The BootCaT toolkit is made up of multiple Perl programs, where the output of one can be fed into the next, making up a suite that can create a corpus from a set of search terms. Baroni and Bernardini (2004, p. 1) wrote:

> The basic idea is very simple: Build a corpus by automatically searching Google for a small set of seed terms; extract new (single-word) terms from this corpus; use the latter to build a new corpus via a new set of automated Google queries; extract new terms/seeds from this corpus and so forth. The final corpus and unigram term list are then used to build a list of multi-word terms. These are sequences of words that must satisfy a set of constraints on their structure, frequency and distribution.

WaCky

From the article where WaCky is introduced, they introduce several subtasks that they describe as the basic steps to construct a web corpus (Bernardini et al., 2006):

1. Selecting seed URLs
2. Crawling
3. Cleaning the data
4. Annotating the data

These subtasks are still essential parts of constructing web corpora today, with some additional subtasks being introduced. The tools and techniques used to solve them, however, have been where most of the discussion, innovation and development has occurred. In-depth discussion of all subtasks, both original and new, will be done in section 2.3.

WaCky was a continuation of the BootCaT effort, and a number of the corpora built within the early WaCky paradigm used BootCaT for several of the subtasks. As WaCky has been a continuous project with a goal of advancing the field of web corpora for several years, there are differences between corpora constructed within the paradigm, especially between the earliest and the latest.

WaCky and its toolset(s) have been used to create several web corpora, like the 2 billion word ukWaC, itWaC (2.6 billion words), frWaC (1.6 billion words), deWaC (1.7 billion words) and noWaC (700 million words). Compared to traditional corpora, like the British National Corpus, with 100 million words, these are quite large corpora.

SpiderLing

Where WaCky introduced and discussed individual subtasks to achieve the construction of web corpora, Suchomel and Pomikálek (2012) created the SpiderLing toolset which does several of the tasks at the same time (like doing language identification while crawling), all pipelined within the same toolset.

The methods of SpiderLing diverge from WaCky's school of thought pertaining to some of the subtasks. Specifically, the crawling is done differently and with a different focus, where a larger effort to make the crawling efficient (not needing to download as much data, while still not losing much of the final corpus data – see section 2.3.1) is made. The data cleaning sub-tasks – boilerplate and duplication removal – are different as well.

COW

Schäfer and Bildhauer (2012)'s *Corpora from the Web* (COW) follows the WaCky initiative's subtasks, but aims to make "incremental improvements" on WaCky's methods and solutions. The result is the *texrex* tool chain, which has been used to construct several COW corpora (including a 16 billion word English corpus and 20 billion word German corpus). The paper fleshes out some of WaCky's descriptions of subtasks to be more in-depth, and introduces their own philosophies and priorities as well.

This can be seen in the crawling, that, while done similarly, has more focus on mitigating *host bias* (see section 2.3.1), as well as almost every step of the data cleaning, which use WaCky's methods as a basis, and then either goes another way for some tasks or expands on the method for others.

Like Wacky, the texrex toolchain has also seen continous development since its inception, with improvements made to crawling (Schäfer, 2016c), text quality assessment (Schäfer, Barbaresi, and Bildhauer, 2013) and boilerplate classification (Schäfer, 2016a).

WebCorpus

Biemann et al. (2013) introduce WebCorpus, a tool chain written in Java and based on the MapReduce framework Hadoop. It also follows many of the same subtasks as COW and WaCky, but implements them in a MapReduce context, due to the nature of the problem being "embarrasingly parallell".

2.3 Techniques and tasks

The problem of constructing web corpora can be split into a number of subtasks. There are variations from web corpus to web corpus and between methods when it comes to how much of a focus they have on a given subtask – some subtasks are almost not mentioned for some corpora at all. As there are no established standards or agreements on which subtasks that make up the task, nor what constitutes those subtasks, this is not that surprising. Common for all web corpora approaches, however, is that they gather data – which I describe in section 2.3.1 below – as well as manage, mold or clean the data, which I discuss in section 2.3.2.

2.3.1 Data collection

To construct a web corpus, one needs data. A common way of collecting this data is by *crawling*. A crawler is programmed to visit web pages and follow links from those web pages to other web pages, and then repeat the procedure recursively. What the crawler does at each web page varies for each use case, but for web corpus construction we want to download the page to later extract the relevant information for use in our corpus.

Data collection by search engine

The BootCaT toolset (Baroni and Bernardini, 2004) does not use its own crawler, but relies on commercial search engine's crawls. This is done by querying these commercial search engines with relevant search terms, extracting text from the top *k* hits, retrieve relevant terms from those texts and repeating the process with these new terms as input. This way of collecting data can give usable results, as seen from (Baroni and Bernardini, 2004), but is impractical when the goal is to construct very large corpora, according to Schäfer and Bildhauer (2012).

DIY crawl

Custom crawls are used by for instance Schäfer and Bildhauer (2012), Baroni, Bernardini, Ferraresi, and Zanchetta (2009), Suchomel and Pomikálek (2012), and Biemann et al. (2013) and crawling seems to be the de-facto norm in modern web corpus construction. Many use the *Heritrix crawler*[1], and customise it to achieve the goals they deem important. The first common step to crawling is the selection of *seed URLs*.

Seed URLs The seed URLs are the URLs given to the crawler as input. They are the starting point of the crawler, and greatly influence the results of the crawl (Ueyama, 2006).

Baroni et al. (2009) choose seed URLs from search engines with a focus on ensuring variety in content and genre. They do this by querying search engines with words from three groups, that give them three lists:

1. Mid-frequency content words from the British National Corpus (not function words).
2. Randomly combining words from the spoken word section of the BNC.
3. Words generated from a word list for foreign learners of English.

[1]http://crawler.archive.org/index.html

They then concatenate these lists. The wish is that they with this will get content from respectively ‘public sphere’ documents, ‘personal interest’ pages, and ‘academic texts’.

Schäfer and Bildhauer (2012) use a similar technique to Wacky’s first mid-frequency content words, and query search engines using 3-tuples of randomly selected content words ranked 1001st through 6000nd in frequency lists. They use a maximum of 10 URLs per 3-tuple.

Host bias An objective for Schäfer and Bildhauer (2012)’s crawling is the notion of reducing *host bias*. While Baroni et al. (2009) have a goal for “the corpus to be representative of the language of interest”, Schäfer and Bildhauer (2012, p. 2) state that

> 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.

The host bias is defined as *the proportion of the total URLs that come from the n most popular hosts in the whole sample*.

By using what they describe as “an extremely large seed set of unique seed URLs”, as well as crawling for an extended period of time, they end up with web pages in their corpus not even indexed by a commercial search engine, and the host bias is shown to go down considerably.

One might wonder if the extension of crawl time helps with decreasing host bias in the general case, as a prolonged crawl does not necessarily mean an increased amount of time not spent crawling the already over-represented host(s). While Schäfer and Bildhauer (2012)’s experimental approach shows a relation between crawl time and host bias, they also point out that this alone is not enough, and that the breadth-first search algorithm itself might need to be revised.

As the Heritrix crawler uses a breadth-first search algorithm, there have been several calls for creating new crawling tools based on random walks. One such crawling tool made by Schäfer (2016c) is ClaraX.

A limit on the number of documents from one host might also be considered, but is not recommended by Schäfer and Bildhauer (2012).

Yield rate While using the Heritrix crawler, Suchomel and Pomikálek (2012) observe that some domains give them almost no resulting corpus data, even though they have downloaded a lot of data from the domain. This is because of the cleaning and filtering of the data – for instance will a Swedish site not be relevant for an English corpus, and be discarded. Because of this, they came up with the notion of the *yield rate*:

$$y = \frac{\text{final data}}{\text{downloaded data}} \tag{2.1}$$

This observation made them move away from the Heritrix crawler, and make their own crawler that samples each domain, decides the yield rate and does not crawl domains with a sample yield rate below a set cutoff.

Schäfer, Barbaresi, and Bildhauer (2014) are able to use something similar in spirit to the yield rate with the Heritrix crawler by crawling for a while, doing their cleaning, and using the 'cleanest' URLs as seeds for the next crawl. This process is then repeated.

Common Crawl

The Common Crawl Foundation is a non-profit organisation that aims to provide crawls of the web free of charge, with the stated goal of "democratizing the web by producing and maintaining an open repository of web crawl data that is universally accessible and analyzable".

The foundation releases (with some exceptions) monthly crawls, made available from Amazon Web Services (AWS)' servers as part of the Public Dataset program. By enabling access to public data for free, AWS states that they hope to "enable more innovation, more quickly". As these data sets are very large, they are a natural point of interest for web corpus construction workers. How to retrieve the data, and how the data is structured is described in more detail in chapter 3.1.

Only Luotolahti, Kanerva, Laippala, Pyysalo, and Ginter (2015) and Schäfer (2016b) have started experimentation on using the Common Crawl to construct corpora, as far as I know.

The potential is certainly there, as a quick comparison of the number of documents between *one* Common Crawl snapshot, and for instance DECOW (before any filtering is done) is telling: The June 2015 Common Crawl snapshot contained 112,700 million documents (Schäfer, 2016b), while DECOW 2012 contained 130 million documents.

While both COW and WaCky select seed URLs for their crawler based on the task of corpus construction and the Common Crawl does not have corpus construction in mind, the sheer size of the Common Crawl makes it an interesting avenue for further corpus construction research.

No matter how the web data is obtained, textual and linguistic content must be extracted and cleaned.

2.3.2 Cleaning the data

The web is noisy, and has a lot of non-relevant data. A website can contain a lot of other things than text, like scripts and markup, and the text that actually is there might not be interesting for corpus use. Menus, buttons, advertisements and tables can contain text, but will often be in keyword form ("Click here", "Home", etc.). There is an evident need for cleaning crawled data, and this is one of the sub-tasks where a lot of work has been done.

There are differences between corpus construction solutions in how these cleaning subtasks are solved – some solve several subtasks in one process, while others separate them fully.

HTML stripping

Removing HTML tags and scripts is a trivial task (Schäfer and Bildhauer, 2012), where the HTML is parsed, tags are removed, and paragraph breaks are inserted where appropriate. HTML entities, like *&amp;* and *&lt;* ('&' and '<' respectively) are normalised to their unicode glyphs, and a second pass of HTML tag stripping is performed. The second pass is neccessary because the HTML entity normalisation can reveal more HTML tags – for instance can &lt; br &gt; become <br>, which must be removed.

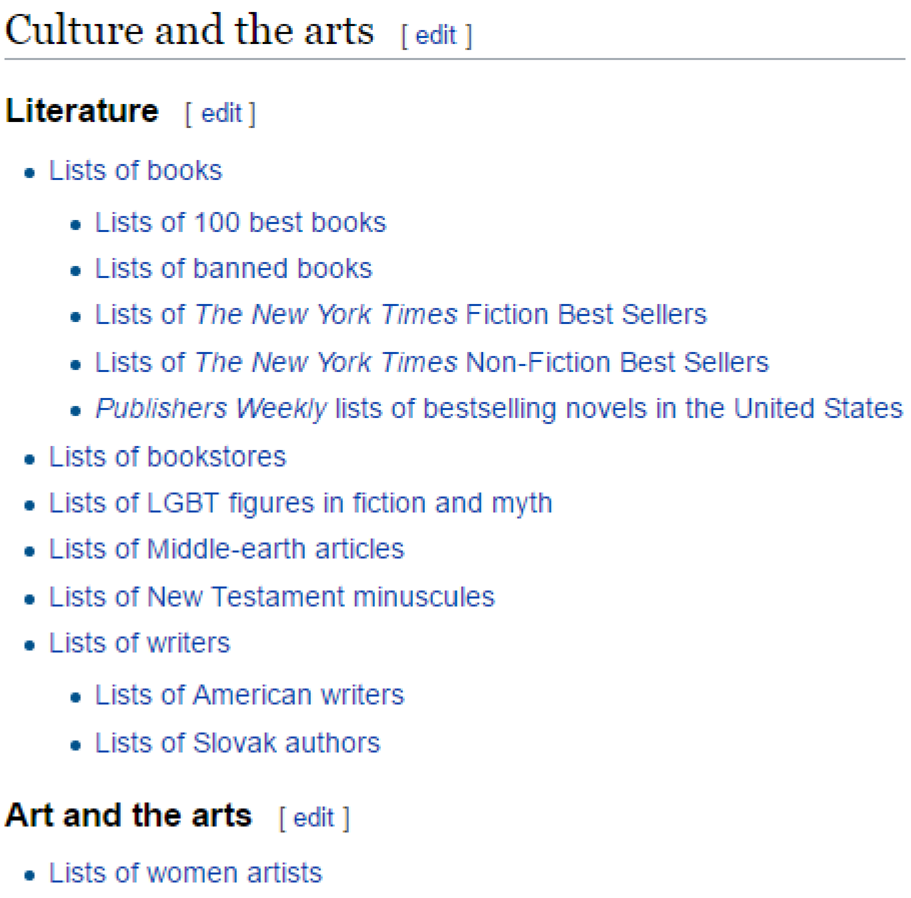
Connected text identification

As we are not interested in documents only containing word clouds, tables, lists or other non-sentences (see figure 2.1) when building web corpora, we have to identify text that is *connected*. 'Connected text' means text that is not a list-like structure of words, but full sentences. Detecting connected text is done while identifying language by both Baroni et al. (2009) and Schäfer and Bildhauer (2012). The latter's methods has since been developed further, and rebranded as "text quality assessment", but still stays true to the main principle (Schäfer et al., 2013).

Baroni et al. (2009) accepts a document if it contains at least ten types, thirty tokens and at least a quarter of all words are function words – closed-class high-frequency words like 'the', 'or', 'and', 'of', 'for'. The accepted document will then be deemed to have enough connected text to be relevant.

Schäfer and Bildhauer (2012) claims this method relies too much on overall document size, and introduces a similar method, but weighted with respect to document size.

This method is again improved upon by Schäfer et al. (2013). Here, the ten most frequent types of a sample of the corpus will be calculated, together with their respective frequencies' weighted (on document length) mean and standard deviation. Then, the more a document deviates from this mean and standard deviation, the worse the document is considered. If the n types together contribute to a document's *badness* past a certain threshold, the document is removed. Note that while Schäfer and Bildhauer (2012) explicitly used function words and Schäfer et al. (2013) do not, the latter's method will consist of mostly function words, regardless, as these often are the most frequent.



Culture and the arts [edit]

Literature [edit]

- Lists of books
    - Lists of 100 best books
    - Lists of banned books
    - Lists of *The New York Times* Fiction Best Sellers
    - Lists of *The New York Times* Non-Fiction Best Sellers
    - *Publishers Weekly* lists of bestselling novels in the United States
- Lists of bookstores
- Lists of LGBT figures in fiction and myth
- Lists of Middle-earth articles
- Lists of New Testament minuscules
- Lists of writers
    - Lists of American writers
    - Lists of Slovak authors

Art and the arts [edit]

- Lists of women artists

Figure 2.1: An excerpt of the Wikipedia page "List of lists of lists", which is an example of an unwanted document for corpus purposes

Language identification

When crawling the web, there is no knowing what language the page you arrived on is in. Although there are indicators, like a higher probability of a language from its country's top-level domain(s), this is no guarantee. More pages from the target language will be retrieved if the words used to retrieve the initial seed URLs all are in said language. Additionally, both HTTP headers and HTML headers allow announcing a document's language, but these fields can both be missing and wrong.

As the types, tokens and function words used as metrics in the connected text evaluation belong to one given language, the language will also be identified during this process. The reason is that the connected text identification will not find the function words it has been given, because the languages differ, and will remove the document.

For instance, searching a Norwegian text for the English words "the" or "and" will not provide the correct frequency counts.

This method can struggle with documents where there are more languages in one site, or where two closely related languages share function words (like Danish and Norwegian). In those cases, another state-of-the-art language identifier might be needed.

Schäfer et al. (2014) use an off-the-shelf naive-bayes classifier to identify languages (Lui and Baldwin, 2012).

Boilerplate removal

Boilerplate typically refers to menus, buttons, labels of input fields, copyright disclaimers, advertisements, navigational elements, etc. Schäfer (2016a, p. 2), provide a more formal definition of boilerplate:

> I define boilerplate as all material that remains after markup stripping, and which does not belong to one of those blocks of content on the web page that contain coherent text.

Using this definition, everything is boilerplate, except if it is related directly to connected text.

Boilerplate removal is one of the areas that are deemed most difficult, and one of the most important to ensure the quality of web corpora. Therefore, a lot of work has been devoted to this task, and I will grant the discussion of this sub-task and its development more space than the others.

In 2007, the CleanEval shared task was initiated, which is a competitive evaluation of cleaning web pages. From this competition, Baroni, Chantree, Kilgarriff, and Sharoff (2008, p. 1), stated:

> It is a low-level, unglamorous task and yet it is increasingly crucial: the better it is done, the better the outcomes. All further layers of linguistic processing depend on the cleanliness of the data.

To solve this, Baroni et al. (2009) lean on the observation that boilerplate is often accompanied by a lot of HTML tags. Of all possible spans of text in the document, they pick the text that has the highest $N(tokens) - N(tags)$.

Schäfer and Bildhauer (2012, p. 4), states the following about this method:

> 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.

CleanEval provided data sets and tools to evaluate boilerplate removal algorithms with. Several general-purpose HTML web page boilerplate detectors using different types of standard machine learning algorithms have been suggested. For instance:

Bauer et al. (2007) utilised Support Vector Machines implemented with "linguistic features" (such as text, token, type or sentence lengths, the frequency of certain keywords that are indicative of clean or dirty text, etc.), "structural features" (density of tags and attributes and whether the tags are indicative of text), and "visual features" (where they render the page and analyse the image). They achieve an F-score[2] of 0.652.

Spousta, Marek, and Pecina (2008) achieved an F-score of 0.8 using Conditional Random Fields with features divided into markup-based features (what kind of tag surrounds or is contained within a paragraph), content-based features (lengths and counts of sentences, tokens, etc. in a paragraph) and document-related features (token and sentence count of the whole document, and the position of the considered block within the document).

A semi-supervised approach using a Naive Bayes Classifier on each token to consider it "out-of-article" (boilerplate) or "in-article" (non-boilerplate), and then finding the highest-scoring sequences of tokens was used by Pasternack and Roth (2009) to achieve a CleanEval F-score of 0.92. The two features used for the Naive Bayes classifier on the CleanEval data was unigrams and the most recent unclosed HTML tag.

Kohlschütter, Fankhauser, and Nejdl (2010) combine techniques – decision trees and linear support vector machines – using 67 features. They identify that using only 2 of them, link density and "text density" (based on number of words and number of lines per 'block' (paragraph)), give almost the same performance as using all features. They achieve an F-score of 0.95.

Suchomel and Pomikálek (2012) used the tools developed by Pomikálek (2011)[3]. Unlike the other methods, jusText does not use machine learning, but rather an algorithm using stop word density, link density and block length as inputs, as well as the context of the block as a cue. The F-score, depending on dataset, is between 0.92 to 0.98.

[2]Unless stated otherwise, I mean F_1-score when I say "F-score".

[3]I was not able to find and read this thesis, so my knowledge of jusText is based on Schäfer (2016a).

Technique	F-score
SVMs	0.65
CRFs	0.8
Naive Bayes	0.9
Combo	0.95
jusText	0.95

Table 2.1: Boilerplate classification techniques evaluated using CleanEval

For texrex, Schäfer and Bildhauer (2012) decided to train a multilayer perceptron with nine different input values, and use it on every paragraph. They end up with an F-score of 0.75 (however, this is not evaluated using the CleanEval evaluation – see below).

Although this was far from state-of-the-art, multilayer perceptrons are very efficient. Therefore, Schäfer (2016a) takes the MLPs in texrex further. By using 37 input features, the F-score is increased to between 0.977 (for English) and 0.994 (for French).

Below is an excerpt of some of the features that were used for training the MLP. While they might not seem to be language dependent, they prove to be, probably due to innate differences between the languages.

- Paragraph length
- Proportion of HTML markup to all text in the non-stripped document
- Number of sentences in paragraph
- Does the paragraph end with punctuation?
- Average sentence length
- Number of sentences ended in punctuation
- The proportion of HTML markup to text in the neighbouring paragraphs
- The proportion of the number of paragraph characters to the whole page
- Is the tag surrounding the paragraph a <p> tag?

It should be noted that the evaluation scores for the texrex boilerplate detection are not evaluated using the CleanEval data set, and is difficult to compare directly with other solutions because of this. The reasons are manifold (Schäfer et al., 2014):

- The CleanEval shared task requires the detection of text structure, like headlines, lists, etc. on top of the detection of whether text is boilerplate or not. This detection is not needed or supported in texrex.

- The data used for evaluation in CleanEval has been stripped of some key data, like HTML and crawl headers, which is used directly as features in the texrex boilerplate classifier.
- The encodings of CleanEval's gold standard data is not normalised to UTF-8 like texrex's data is, and cannot be directly matched by the CleanEval evaluation script.
- CleanEval's annotation guidelines are designed for destructive removal, while texrex is designed for non-destructive classification. This difference in philosophy makes for some incompatible differences: Some content that *is* boilerplate, but is relevant or important at the same time must be classified as non-boilerplate by CleanEval, so it does not get deleted. texrex can keep it *and* deem it to be boilerplate. An example of such content is non-human generated timestamps for forum posts. They give important information, but is definitely boilerplate.
- CleanEval broke new ground in 2008, but technology moves fast, and 9 years is a long time. The web looked different then and HTML5, for instance, didn't even exist. Even though CleanEval evaluation is still used by some developers of corpus creation tools (Habernal, Zayed, and Gurevych, 2016), it can be argued that it is outdated at this point.

See section 2.5.2 for developments that have happened during the writing of this thesis.

Duplicate and near-duplicate removal

Duplicates occur quite frequently on the web. Whether it is the same quote, book or article that is quoted on different pages, or actual true copies, they need to be dealt with to avoid a doubling of a document's effect on the token frequencies.

True duplicate removal True duplicates are the documents that are identical.

To remove these, Schäfer and Bildhauer (2012) create an array for each document, and fills it with evenly distributed characters in the document. If two such arrays match, the copy is discarded. Suchomel and Pomikálek (2012) hash the documents in two steps, both before they were cleaned and afterwards, then remove all matching copies.

Near-duplicate removal Documents can also be near-duplicates, where a lot of the content is found in other documents.

Doing this step after the boilerplate removal can be beneficial, as boilerplate can create false positives (by different content sharing so much boilerplate that the duplication removal algorithm thinks they are identical), and false negatives (the same content having different boilerplate, making the classifier think they are different documents) (Baroni et al., 2009).

Both Baroni et al. (2009) and Schäfer and Bildhauer (2012) use variations of a w-shingling algorithm as described by Broder (1997). In essence, the algorithm makes n-grams – called shingles – out of the document's tokens and sees how many of these are shared between other documents to find how similar they are.

With n = 4, the document "a shingle is a shingle is a shingle" can be made into shingles like this:

Document: "a shingle is a shingle is a shingle"
shingles: { (a,shingle,is,a), (shingle,is,a,shingle), (is,a,shingle,is) }

The first two shingles will show up twice. For efficiency, these shingles can be hashed. If the shingles that were shared between two documents exceed a treshold of resemblance (using a *Jaccard coefficient*, for instance), the longest is kept. Baroni et al. (2009) use a simplified version of the algorithm, where the shingles are not hashed like described by Broder (1997). Additionally, they only perform near-duplicate removal, and does not perform perfect duplicate removal.

Suchomel and Pomikálek (2012) use a tool called onion[4], that performs deduplication on a paragraph level. Paragraphs containing more than 50% 7-tuples previously encountered are discarded.

Encoding

The web contains a lot of pages, with many of them using different encodings. With the limitations from the ASCII standard of early computers making people from different regions and countries come up with their own encoding to be able to use their own characters, a corpus must accomodate for this encoding diversity. Additionally, a corpus should unify the encodings so all the documents have the same encoding throughout the corpus.

The HTTP protocol has ways of announcing the encoding a page is using, but there are several ways this can be misleading. There might be text from different encodings in the same page, the text might be corrupted and not follow any encoding (or the wrong one), or the encoding announced might be different from the actual encoding on the page.

[4]http://corpus.tools/wiki/Onion

Web browsers are robust when it comes to encoding, being able to handle most of these cases, and the corpus must be able to as well.

The obvious encoding for a modern corpus to use as its standard is unicode (UTF-8), the encoding trying to unify characters from as many other encodings as possible.

2.3.3 Metadata and annotation

Metadata about each document can be important information depending on what the corpus is to be used for. Document metadata can often be found in the HTML headers of a document in "<meta>"-tags or in the HTTP headers, where information such as what type of content the page contains, keywords to describe the content, what language it is in, etc. can be saved. Whether to either use this data while constructing the corpus or making the metadata available to the corpus user in some fashion are decisions corpus constructors must make.

While all kinds of metadata can be saved in the HTML headers, normalising these can be difficult, as some documents may have little to no metadata and the metadata that exists may be saved differently, or have different notation. Some linguists might be interested in for instance the gender, age or nationality of the author of the text, and those kinds of data will often be approximations, often through automatic classification (Biemann et al., 2013).

Annotating the corpus means performing tasks such as tokenisation, lemmatisation and part-of-speech tagging. Some libraries and software seem to be quite commonly used for these tasks – often off-the-shelf software. One can question whether this software is state of the art, and how precise the results of the off-the-shelf software are.

2.4 Evaluating web corpora

Like mentioned earlier, Banko and Brill (2001) make the case that larger corpora improve the performance of machine learning algorithms more than manually tuning the algorithms themselves. Intuitively, however, size is not everything: For instance, having the same word or sentence repeated over and over will not get you desired results. Additionally, Versley and Panchenko (2012) give several examples where using the manually annotated 100 million token British National Corpus give superior results to using the 2.25 billion token ukWaC.

Therefore some notion of corpus quality must be assessed. There currently seems to be two quite different approaches to this challenge.

One approach is to assess the given corpus' quality by finding ways to compare it to other corpora. By comparing corpus statistics, on top of corpus size, one can say something about the corpus quality – and get hints to what one might need to improve: Did we identify any languages other than our target language? Did we manage to remove duplicates? Did the boilerplate removal remove "sufficient" boilerplate? etc.

Another approach is the linguistic evaluation of the web corpus as a representative sample of the language in general.

2.4.1 Statistical quality metrics

Statistical parameters can be used to characterize a corpus – and thus enable comparing corpora.

According to Biemann et al. (2013), typical assessment statistics for corpus quality are for instance the distribution of word, sentence or document length, the distributions of characters or n-grams, or the corpus' agreement with certain empirical laws of language such as Zipf's law.

These statistical measures can be used to identify anomalies, or signs that a document should have been caught by the filtering procedures. For instance, Biemann et al. (2013) introduce several statistics that they use to identify different kinds of preprocessing problems:

- The relative size of the largest domains represented in the corpus can indicate problems with the crawling, as it can be an indicator of host bias.
- The length distribution of sentences can be used to identify problems with near-duplicate removal: If the length of the sentences doesn't follow the same distribution as expected (for a given language, genre, etc.), there might be sentence duplication affecting it. An example is in figure 2.2, where you can see heavy spikes for Hindi news. Biemann et al. (2013) say these showed signs of being, and proved to be, boilerplate material and duplication that should have been removed.

2.4.2 Do web corpora represent languages as well as traditional corpora?

In their WaCky-article, Baroni et al. (2009, p. 5) state:

> Notice that the rationale here is for the corpus to include a sample of pages that are representative of the language of interest, rather than getting a random sample of web pages representative of the language of the web.

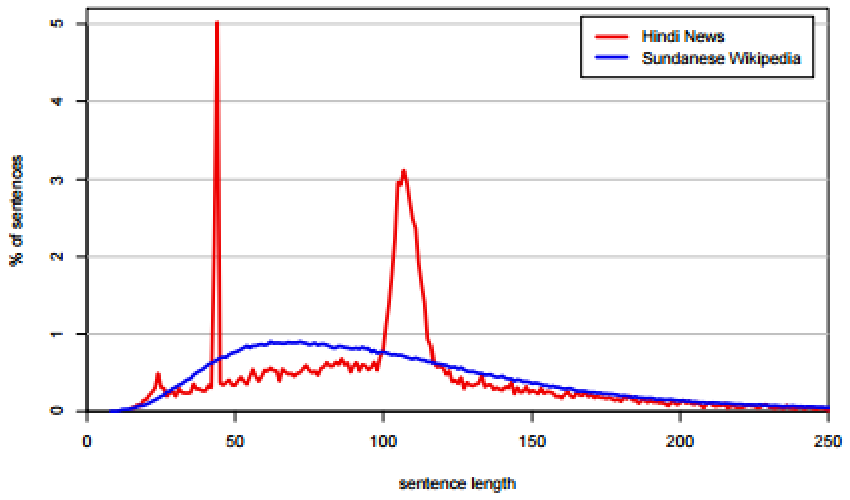


Figure 2.2: Example of how assessing corpus statistics enables comparing corpora and disovering signs of flaws

Herein lies the notion of *representativeness* – the notion that the documents you retrieve represent some bigger whole. The quote from WaCky even makes clear that they take steps to make sure their result *is* representative of something, and also that it *is not* representative of something else. They do this by detecting text types and topics while crawling, and making sure that their corpus includes documents from newspapers, spoken word, technical manuals as well as blogs and forum posts. By making sure it does, Baroni et al. (2009) seem to believe their corpus to be sufficiently representative.

"As representative as reference corpora"

Biemann et al. (2013, p. 15) emphasize the syntactic and lexical core of the language as what to base the evaluation of the corpus' representativeness on:

> The linguistic usefulness of web corpora as representative samples of general language (rather than web-specific genres) can be evaluated by comparison with a traditional reference corpus, using frequent general-language words and constructions as test items. The underlying assumption is that an ideal web corpus should agree with the reference corpus on the syntactic and lexical core of the language, while offering better coverage of less frequent words and construction [sic], highly specialized expressions, and recently coined words.

That is: If a web corpus and a manually constructed reference corpus agree on core parts of the language, the web corpus is as representative of the language as the reference corpus – which is commonly believed to be sufficiently representative. Thus, the web corpus can also be said to be sufficiently representative.

Biemann et al. (2013) also offer examples of how to test this agreement on language core, by using the corpora for two tasks, and comparing the corpus performance on those tasks: the first task were verb–particle combinations for English, and the second creating word collocation lists based on the corpora.

The results from the first task show that the manually constructed British National Corpus does well, but that the web corpora do better as their size increase. However, the good performance of the web corpora requires more than a magnitude more data than the BNC. Interestingly, the very large Web1T5 (Google's 1TB large 5-gram database) performs very poorly, in spite of being much larger. This can be a sign of poor quality or its limited context, and further emphasizes the need for both high quality, and large corpora.

The same linear performance increase with size is seen for the second task, but here the BNC outperforms the web corpora, although not by much. This shows that web corpora for some tasks can do as well as reference corpora, and as their size increase, perhaps surpass them.

Representative of the general language

A wish for some corpus users is to be able to say something general about the language as a whole by being able to say something about the corpus. For this to be true, the corpus must be representative of that language. This is a field of some controversy – for instance, Kilgarriff and Grefenstette (2003) say that it is impossible to create a corpus representative of English, as the population of "English language events" are so difficult to define, and to know sufficiently about to create something that is representative of it.

For the corpus to be representative, or balanced, two things should be evaluated, according to Biemann et al. (2013):

1. Is the corpus a "random selection" of the population – i.e. the language? If so, it should reflect the distribution of topics, genres, text types, etc. However, if one managed to gather a truly random sample of the web in a given language, is that the same as getting a truly random sample of a given language?

2. No genre or topic should be heavily over-represented – it should be "balanced". If one imagines that the web does not have the same distribution of genres or topics as what the distribution would be in a reference corpus, is a random selection of the web desirable if wanting balance?

While Biemann et al. (2013) avoid giving a recommendation on whether to prioritise randomness or balance while constructing corpora, Schäfer and Bildhauer (2012) seem to favour the random selection approach, stating that they do not believe a corpus could ever be balanced.

2.5 The goal of this project

This project will investigate how one can use the Common Crawl's data to construct a large web corpus. There are several diverging paths depending on what is found from these investigations and what decisions are taken based on those.

2.5.1 High-quality connected text extraction from the Common Crawl

In addition to the raw archive files (WARC), Common Crawl offers extracted text versions of their crawls, so-called WET-files. An initial task will be to investigate how viable these text extractions are for corpus construction with regards to quality.

The results of that investigation will determine what files to proceed with, which will in turn affect the corpus creation process.

Finally, when a corpus has been constructed, its usefulness needs to be evaluated – how that will happen will be discussed in chapter 5.

2.5.2 Concurrent work

C4Corpus

While this thesis was being written, other relevant work on the topic has been occuring at the same time. Some of it I might have been able to integrate with this thesis, while other papers or toolkits might have been released too late in the process to be accomodated for.

One such toolkit is the C4Corpus[5] (Habernal et al., 2016).

Like this thesis, they use Common Crawl data as the basis for their constructed corpus, but diverge from the choices made in coming chapters in other areas. Their software is written in Java, using the Hadoop framework, and they focus heavily on *licensing* – making their corpus available to the public for free, by restricting their corpus to documents with a CreativeCommons license.

Their boilerplate removal method is a Java reimplementation of jusText, the same library used in the earlier mentioned paper by Suchomel and Pomikálek (2012).

[5]https://github.com/dkpro/dkpro-c4corpus

The deduplication is similar to the one popularly used as perfect-duplicate removal, the hashing of 64-bit shingles then used as fingerprints for documents.

CleanerEval

As a follow-up to 2008's CleanEval, the 2017 Web as Corpus Workshop, hosted by the SIGWAC will feature the first meeting of a new shared task: CleanerEval. In the same spirit as Schäfer (2016a) and Schäfer et al. (2013), the shared task will focus on boilerplate removal and assessing the linguistic quality of a document.

The WAC-XI conference will be hosted July 27, 2017.

Chapter 3

Exploring the Common Crawl

Recall the description of the Common Crawl in section 2.3.1. The data is available both for download, or to be used directly by running software on Amazon's servers – either with their EC2 instances or their Hosted Hadoop clusters. In this project, I will be downloading the data to NorStore – the Norwegian national infrastructure storage resource, and use it with the University of Oslo's cluster Abel (see section 4.2).

3.1 The data

Common Crawl data is stored in three different ways, as:

- Raw crawl data
- Metadata
- Extracted text data

WARC The raw web data (with the HTTP header information included) is provided in the Web ARChive format – or the WARC format. This format is an ISO standard format building on the ARC format orginally used by The Internet Archive to archive web pages (International Organization for Standardization, 2009). The WARC files contain four types of entries:

- The WARC info-entry: There is one of these per WARC file describing the WARC format, the date of its saving, etc.
- Requests: The HTTP requests that were sent.
- Responses: The HTTP responses, including headers.
- Metadata: Computed metadata entries – for the August 2016 crawl, this only contained the fetch time of the crawl.

Type	# files	cmpr. file size	dec. file size	cmpr. rate	cmpr. time	entries per file	docs per file	doc size
WARC	29,800	988	4515	3	33s	155,400	51,800	0.10
WET	29,800	156	405	2.6	6.7s	51,800	51,800	0.01
WAT	29,800	353	1524	4.3	13.6s	155,400	N/A	0.01

Table 3.1: Data collected about the WARC, WET and WAT files from the August 2016 crawl. All the data (except the first column) are averages from a random selection of 100 files – where the WET and WAT files selected are the ones derived from the WARCs. All file sizes are in megabytes. The rows are: Number of files, compressed file size, decompressed file size, compression rate, compression time, number of entries per file, documents per file and the size of each document.

Each entry has additional meta information prepended to them. The information provided in these *WARC-headers* include the date of the response, the unique ID of the record, the length of the content, what type the content is, the URI of the target, as well as the IP address of the target. The Common Crawl web site provides code snippets for use with several different programming languages to get started with the WARC files[1], and the Internet Archive also provides libraries to retrieve metadata and the documents themselves[2].

WAT The metadata from the WARC files, that is computed for each request/response entry can also be found in the WAT files – the Web Archive Transformation format. The WAT entries contain metadata from the WARC-headers, the HTTP request and the response. If the response is in the HTML format, the HTTP headers and the links on the HTML page are made available as well. This is then stored in a key–value Javascript Object Notation (JSON) format.

WET In the case of this project, I am interested in the textual data of the crawls, and the Common Crawl provides extracted plaintext for each response entry in the WARC files. This is stored in the WET format (Web Extracted Text), where the WARC headers and information about the length of the plaintext data are prepended to the actual extracted plaintext.

Only the entries for which there exists content that it is possible to extract text from is in these files. This means that the request and metadata entries are stripped, as well as content where the content-type of the response is considered to not be textual.

[1] http://commoncrawl.org/the-data/examples/

[2] https://github.com/internetarchive/warc

Operation	Time required
Reading	17.5 hours
Decompressing	11 days
Downloading	14.5 days

Table 3.2: The time required for performing different operations on the August 2016 Common Crawl

As the size of each crawl is quite large, the data is split in a large number of gzipped files containing thousands of WARC entries. For the August crawl of 2016, the file count for the WARC files was 29 800, where each file was about 1GB (see table 3.1). As the WAT and WET files are generated based on the WARC files, the file count is similar for those, but while the compressed WARC files are sized around 1GB, the WET and WAT files are a lot smaller, with WARC >> WAT > WET.

While it seems obvious that the original documents in the WARC files take up the most space, it might not be as intuitive as to why the files only containing metadata, the WAT files, are much larger than the extracted text files, the WET files. The reason for this is that the WET files only contain records where there are textual content to be extracted, and where no HTTP header information is included. Subsequently, the WAT files contain information about both the request and the response, while WET only contains responses.

A master paths file for each of WARC, WET and WAT is published on the Common Crawl's Amazon bucket, as well as with a link from the Common Crawl blog[3]. These paths files contain a relative link to a file per line. Concatenating the relevant prefix with the line results in the absolute link needed to access the file. The prefix can either be in http form or in S3 form, depending on whether you want to access the data through HTTP or through Amazon's Simple Storage Service protocol.

3.1.1 A note on scale

30 terabytes of web archive data is a lot of data. Both handling and processing data of this scale takes a lot of time. As an example, the time required to just open the decompressed archive files, and reading through them without any actual processing took about 17.5 hours. Any additional operation performed while processing the data will only increase this amount of time. See table 3.2.

To be able to analyse and process the data in a more reasonable amount of time, parallel computing is an absolute neccessity. Parallelising the processes is not

[3]http://commoncrawl.org/connect/blog/

always trivial, and adds additional development time on top of the time already required.

In addition to the amount of time required to process the complete crawl, each file is also of significant size. A compressed WARC file at 1 GB can require from seconds to a couple of minutes to process, and makes the process of creating and testing analysis tools even longer.

The scale of the data affects the whole process, from data analysis to the actual processing of it for corpus creation, and should be taken into account when planning these types of projects.

3.1.2 Retrieving the data

As retrieving about 30 000 1GB sized WARC files can take some time, parallelisation of the download process can seem like a time-saving prospect. However, this requires that both the client and the server can handle large parallel data transfers. In my case, this means that NorStore needs to handle parallel transfers well. Consequenctly, NorStore needs to have a good bandwith, that the file system can be saved to in parallel, and that the processing power is great enough to handle saving to the file system. Additionally, parallelisation is also dependant on Amazon's routing: If two separate download requests from the same client go to two different Amazon end points, the efficiency of the transfer is maximimised.

In a preliminary study, downloading 150 files, using 1, 2, 4, 8, 16 and 32 processes at the same time (but on a CPU with only 16 cores), gave the results shown in figure 3.1.

The results show that parallelising using 32 processes cuts the download time to 13.3% of the non-parallel download time, from an average download speed of 23.9 MB/s with one process, to an average download speed of 200.5 MB/s with 32 processes. That makes the total download time of the complete 30 terabytes about 41 hours. Compared to the sequential download speed of 14.5 days, this is a big improvement.

As seen from the graph, the relative speedup gain is the largest up to 8 processors, but the gain with 16 processors is still good, although not as efficient as 8 – see the efficiency comparison below. I also tried using 32 processes, and did receive a small speedup, although arguably negligible. As the CPU did not have more than 16 cores, this is not that surprising.

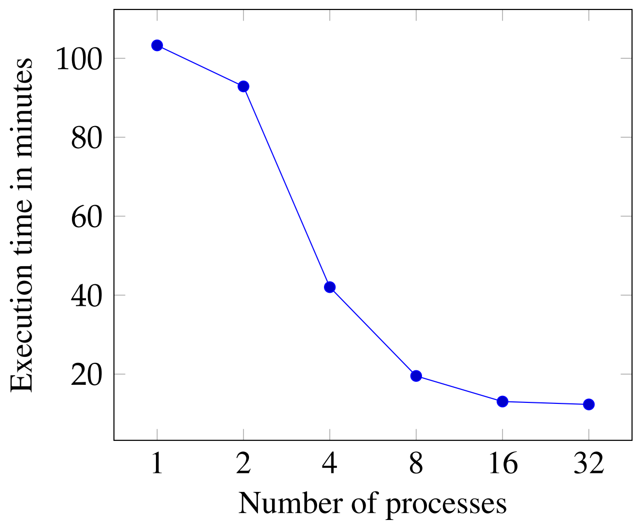


Figure 3.1: The execution time of downloading the same 150 gzipped WARC files per number of processes. (Note that this was done on a CPU with 16 cores)

Processors	Speedup	Efficiency
2	1.1	**0.6**
4	2.5	**0.6**
8	5.3	**0.6**
16	7.9	0.5
32	**8.4**	0.3

Table 3.3: The speedup and efficiency of the download using a different number of processors

Above one can see that going from 8 processes to 16 does give a speedup (5.3 to 7.9), while the efficiency goes down (0.66 to 0.49) – each process isn't used as efficiently as possible.

3.2 Getting my feet WET

As the crawls already offer extracted plaintext as WET files, these files are a natural place to start: If the text extraction is of sufficient quality, there is no need to do such an extraction step myself.

While remaining HTML in the WET files can be fixed in an additional HTML stripping step, there is little point in using the WET files as the basis if the reason we wish to use them is to forego this step entirely.

3.2.1 A preliminary inspection

An initial rough inspection of one WET file reveal that quite a lot of HTML remain in the documents. By picking one WET file at random, splitting it up to one file per crawled document, and then doing a simple search for remaining HTML by searching for occurences of "<div>", "<article>", "<span>", "<p>" , "<body>", "<a>" and "<html>", I found the following:

The file contained 50300 documents, where 1300 of them still had HTML remaining. The average number of HTML tags per file that still contained HTML were about 5, and this was with a very naive approach: There are more HTML tags than the ones I searched for, as well as Javascript and CSS that this approach won't find. However, it is enough to see that the HTML is not sufficiently stripped, and that a separate step of text extraction is required. See figure 3.2 for an example of a document where there are still remnants of HTML.

3.2.2 WET vs. cleaned WARC

In chapter 4, I describe my chosen method for text extraction – texrex. Extending the inspection above by comparing the corpus made from texrex with the WET files further cements the need for using the WARC files, and giving up the WET files. Comparing both a corpus made from four WARC files and the corresponding four WET files for remaining HTML tags, remaining HTML entities, encoding errors, token count and language distribution, the corpus outperforms the WET files in every regard.

It is important to note that this test corpus was made with parts like boilerplate detection and language detection turned off, as both are language dependent, and must be switched off to be able to assess the corpus without committing to one language. The size filtering option – where documents with a size above or below certain thresholds are removed both before and after cleaning (see section 4.2.1) – was turned on, however, together with the de-duplication. Because of these filters, there are fewer documents in the corpus than in the WET files.

For the files picked, this ended up being 90 500 documents in the constructed corpus and 244 000 documents in the WET files. Note that this means that about 160 000 documents that were included in the WET files are not included in the corpus. This stood out as a large amount, and is addressed in section 3.2.3.

Below, I retrieve data from the WET files, the corpus made with texrex, as well as the documents from WET that can also be found in the corpus, the intersection (noted as 'WET ∩ corpus' below).

[4] WET ∩ corpus are the documents from the WET files, but where the documents that is not included in the constructed corpus is removed. This makes for a more direct comparison.

Data	tags	# docs w/ tags (σ)	% docs w/ tags	tags/doc
WET	37,173	3055 (5.50)	1.20%	0.2
WARC-corpus	28	28 (0.02)	0.03%	0.0003
WET ∩ corpus[4]	15,114	1139 (5.40)	1.26%	0.17

Table 3.4: The remaining HTML tags of the different corpora. From the left: The number of tags remaining, the number of documents that had remaining tags, the percentage of documents that had remaining tags and how many tags there were per document

Data	entities	# docs w/ entities (σ)	% docs w/ entities	entities per doc
WET	166,307	5311 (55.3)	3.2%	0.7
WARC-corpus	8,772	2284 (1.7)	2.5%	0.1
WET ∩ corpus	136,770	2897 (77.7)	3.2%	1.5

Table 3.5: The remaining HTML entities of the different corpora. From the left: The number of entities remaining, the number of documents that had remaining entities, the percentage of documents that had remaining entities and how many entities there were per document

From table 3.4, it is clear that the WET files are far inferior to the corpus made with texrex in terms of removal of HTML tags. The HTML stripping done on the WET files is lacking severely, while the HTML stripping from the texrex corpus has managed to remove almost every tag.

There are a lot of remaining HTML entities in both the WET files and the corpus. However, compared to the WET files, the corpus has far fewer entities remaining, with the intersection having about 16 times as many entities remaining (see table 3.5). While the corpus is in the XML format, and therefore needs to escape some characters with XML entities itself (namely *&amp;* (&), *&quot;* ("), *&apos;* ('), *&lt;* (<) and *&gt;* (>)), these are not the entities making this number so large, as the top entities in the texrex corpus were dominated by entities not in the XML specification (see table 3.6). All of these entities have corresponding unicode glyphs, so this suggests that the entity normalisation is incomplete.

Entity	Frequency
&nbsp;	4594
&lhblk;	1170
&amp;	510
&copy;	345
&diams;	288

Table 3.6: Top five frequent entities in the texrex corpus

As the WET files do not include the HTTP headers like the WARC files do, one must assume that the encoding is supposed to be unified. However, trying to interpret the documents as unicode showed that this work has not been completed.

Corpus	# docs	% docs
WET	5019	2.1
WARC corpus	0	0
WET ∩ corpus	1990	2.2

Table 3.7: Number of documents with encoding errors

Table 3.7 shows the amount of documents with encoding errors, i.e. documents containing bytes not possible to interpret and convert to unicode.

Corpus	# tokens	#tokens/doc	σ (SD)
WET	226,261,200	927.3	2140.6
WARC-corpus	93,685,600	1035.2	1508.6
WET ∩ corpus	101,109,315	1117.23	2842.3

Table 3.8: Number of tokens in the corpora

The number of tokens per document in the different corpora are quite similar across WET and the constructed corpus (table 3.8), but the differences are interesting: The difference between *WET* and the *WET ∩ corpus* makes it probable that the constructed corpus indeed selected the documents where more text remained after cleaning. The difference between the constructed corpus and the intersection might be because of the remaining HTML in the intersection documents. The standard deviation is larger for the WET files, maybe because of the size filtering that texrex performs, which removes the extremes on both sides.

Note that the tokeniser used is a naive white-space separation, which only approximates the token counts. For some languages, where tokens are not separated by white space, this way of tokenising will result in a token count of 1. As the primary language in the corpus is English, as well as the language distribution being quite similar, this will be an (good enough for this use) approximation.

3.2.3 Where do the documents go?

As stated earlier, what intuitively seems like a lot of the documents from the WARC files did not carry over to the final constructed corpus, and the WET files contain almost triple the amount of documents in relation to the corpus. These documents could contain a non-negligible amount of data, and should be accounted for.

There are several texrex stages where documents are discarded. Although some of them were turned off for this experiment, a breakdown of where documents disappear is still useful, and can shed some light on where the documents for this experiment went.

Schäfer and Bildhauer (2012) provide this table of how many documents that each process removed:

Algorithm	% Removed
Size threshold	71.67%
(Language)	12.93%
Perfect duplicates	2.43%
Near duplicates	7.03%
Total	94.06%

Table 3.9: The percentage of documents removed per processing step from the total number of initial documents. Language detection was not performed for the experiment, but is left in the table for illustration.

These numbers will not be the same for each crawl and each corpus, and will depend on the configuration of the tool. However, it does help to explain why about 60% of the documents from the WARC files in our experiment were gone from the final corpus.

The largest culprit, the size filters, are explained more thoroughly in section 4.2.1.

3.2.4 Language distribution

I also ran language identification on the same files as the above inspection was performed on, to see if there were differences. The language identifier that was used[5] output the top three languages being present in one document, and a number, the percentage of the document that was in said language. The tables below are how many of the characters in the whole corpus that was deemed to be in a given language.

Language	WET	WARC-corpus	WET ∩ Corpus
English	81.4%	75.3%	78.6%
German	2.0%	2.5%	2.0%
Spanish	1.9%	2.5%	2.1%
Chinese	1.5%	3.3%	2.0%
French	1.3%	1.7%	1.2%
...			
Norwegian	0.12%	0.08%	0.08%

Table 3.10: Distribution of language in the different corpora, as percentage of total number of characters

Table 3.10 says that the language distribution is quite similar between the corpus and the intersected WET documents. The differences can both be because of the remaining HTML confusing the classifier, or because some documents couldn't get their language classified because of encoding errors in the WET files.

3.2.5 What now?

The WET files are clearly not cleaned sufficiently to be usable without doing a lot of the steps that I would be doing with the WARC files anyway. Therefore, I will be continuing with the WARC files and leaving the WET files behind.

As the Common Crawl seem dominated by English content and does not contain enough data from under-resourced languages, like Norwegian, to construct a corpus of sufficient quantity, I will proceed with the goal of constructing an English corpus.

[5]https://github.com/CLD2Owners/cld2

```
WARC/1.0
WARC-Type: conversion
WARC-Target-URI: http://1003thepeak.iheart.com/photos/
    snap-shots/celebrity-dogs-366430/
WARC-Date: 2016-08-24T04:56:35Z
WARC-Record-ID: <urn:uuid:bbe19425-884e-48f2-92be-980
    a4270c089>
WARC-Refers-To: <urn:uuid:61370b1a-8107-4084-83e4-
    d483e9045c2d>
WARC-Block-Digest: sha1:
    IUC52ZDJFWFA72ZJMCAEXNC2DA7VNNYL
Content-Type: text/plain
Content-Length: 4492

<p><span style="font-size:␣9px;">Photo: Splash news </
    span> If Jessica Hart was a dog she would be this
    thing.
<p><span style="font-size:␣9px;">Photo: Splash news </
    span> Snooki and JWoww dress to match their dogs.
<p><span style="font-size:␣9px;">Photo: Splash news </
    span> We can't really tell which is Ozzy!
    Incredible!
<p><span style="font-size:␣9px;">Photo: Splash news </
    span> Coco and her pitbull. Identical!
<p><span style="font-size:␣9px;">Photo: Splash news </
    span> Luke Wilson looks a lot like his stocky pup.
<p><span style="font-size:␣9px;">Photo: Splash news </
    span> This is amanda Byne's dog... <p><span style="
    font-size:␣9px;">Photo: Splash news </span> Blondes
    ! Seann William Scott would most definitely have a
    yellow lab.
<p><span style="font-size:␣9px;">Photo: Splash news </
    span> Brunettes! Anne Hathaway with her chocolate
    Labrador, Esmeralda. <p><span style="font-size:␣9px
    ;">Photo: Splash news </span> Jon Hamm and his mutt
     often make similar facial expressions.
```

Figure 3.2: An example extract from what should have been extracted text without HTML remaining from a WET file of the Common Crawl

Chapter 4

Text extraction from the Common Crawl

Given the limitations of the extracted text files offered by the Common Crawl that were established in section 3.2, other ways of producing high-quality linguistic content are needed.

Such a process will need to process WARC files, and by performing tasks as described in chapter 2 – such as for instance HTML stripping, boilerplate removal, duplicate and near-duplicate removal – produce files containing extracted text.

4.1 Choosing a tool for the job

In chapter 2, I investigated solutions to the aforementioned challenges. While earlier efforts such as WaCky (Bernardini et al., 2006) broke new ground when it was released, newer solutions have shown that they give better results. In particular, Schäfer and Bildhauer (2012)'s texrex, Biemann et al. (2013)'s WebCorpus[1] and the individual software in Suchomel and Pomikálek (2012)'s SpiderLing stuck out in my preliminary research as the ones that appeared to be the most modern and state-of-the-art of the alternatives. For our purposes, there are several desiderata that the result should satisfy, as well as others that the process itself should satisfy.

4.1.1 Desiderata

The most important goals for the resulting extracted text to fulfill are:

[1]https://sourceforge.net/projects/webcorpus/

- High quality (relatively “clean”) data
- High quantity data
- Delivered in a format that is easy to use further, easy to extract relevant metadata from, etc.
- Preserving as much information as possible, such as original paragraph breaks, as well as metadata
- Should be in one language
- Should represent that language to a certain degree (i.e. removing non-relevant text (boilerplate) successfully)
- Generation of additional data and annotations is a plus (POS-tagging, tokenising, etc.)

Additionally, the process itself has some desired properties. It should be:

- Relatively easy to setup and use
- Possible/practical to run on the University of Oslo’s architecture
- Should scale well with parallel processing units
- Have a good degree of configurability
- If a process is unsuccessful and terminates prematurely, ease of continuing where it left off
- Ideally cover the whole process, from reading WARC files, to outputting the desired result

texrex and WebCorpus are two different pieces of software trying to solve a similar problem, both fulfilling many of my desiderata. They both read both ARC and WARC files and produce clean, high-quality text from them. WebCorpus computes and generates more useful data as well, like generation of an n-gram corpus and a cooccurence corpus.

SpiderLing, meanwhile, pipeline its crawler with the boilerplate removal of Pomikálek (2011), and the onion deduplication, among others. While using the separate sub-task solutions indivudually is possible, the amount of work required to do so seems substantially higher than the others.

Comparing the subtask solutions of the different tools directly is difficult. Some of the WebCorpus techniques, like the boilerplate detection, I can not seem to find evaluations for at all, and as stated in section 2.3.2, there are differences in how the boilerplate removal is evaluated. The quality of the corpus depends heavily upon the boilerplate detection, and the solution used in texrex probably is the current state of the art (agin, see section 2.3.2), which contributes further to the choice of texrex over the others. It must be said, however, that the jusText algorithm used in SpiderLing also performs very well in evaluations.

In the end, I will use texrex because it is maintained, highly configurable and not, unlike WebCorpus, using the Hadoop framework (which there is no immediate support for at the University of Oslo's Abel cluster). As texrex requires less work than the tools in SpiderLing to set up because of the way it performs almost all the desired tasks while processing the WARC files, this further demonstrates the advantages of texrex.

4.1.2 texrex: An overview

All the COW corpora are output of the tool chain *texrex*, made by Roland Schäfer and offered as free software on github[2]. It is written in FreePascal, and fulfills most of the desiderata mentioned in section 4.1.1: it produces high-quality text from the WARC files that the Common Crawl use, delivered in an easy-to-understand XML format. However, it does not perform additional annotation, such as tokenisation or PoS-tagging, and does not support continuing a task where it dropped off.

[2]https://github.com/rsling/texrex

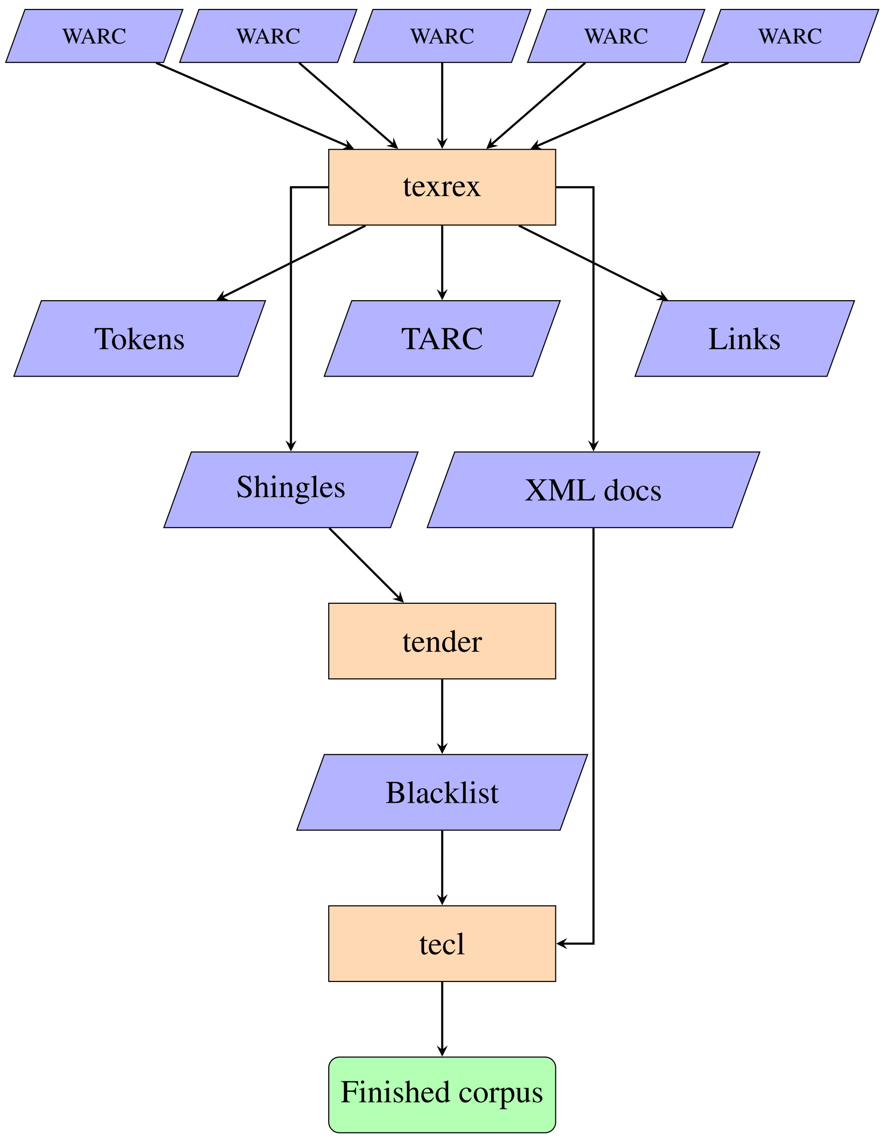


Figure 4.1: The flow of texrex, showing the output of texrex entered as input to later tools

The toolchain is made up of several programs – see below, as well as figure 4.1.2. One typical run will use three of these programs, feeding the output of one into the next: texrex → tender → tecl.

texrex Named like the toolchain, this program is where most of the corpus construction takes place. The input is a configuration file, where the user can configure many of the subtasks of the process, in addition to the WARC files. The user can also specify what types of output are desired, where it is possible to produce the following:

- Shingles – the document fingerprints used to detect duplication in *tender*

- The corpus – an XML representation of the extracted text documents, see figure 4.2 for an example of the output
- TARC – the WARC files used, without the documents removed by texrex's sub tasks
- Tokens – A list of tokens and their frequency per document
- Links – A list of all the links contained per document

This leaves the task of duplication removal, which is performed in *tender* and *tecl*. The shingles and the XML representation of the corpus are used further, the shingles in *tender* and the corpus in *tecl*.

tender Recall the description of w-shingling in chapter 2.3.2. The purpose of the processing steps in tender is to identify documents that exceed a treshold of resemblance. However, the tool does not remove the documents deemed to be near-duplicates, but rather lists them. The tool processes the shingles from one or several texrex runs. It then outputs:

- Sorted shingles – the same shingles that were input, only sorted. Useful when splitting the texrex job in several parallel parts, as the sorted shingles can be used further without needing to sort them again.
- Sorted doc–doc-pairs – all the shingles in every document are listed in sorted document order, to be used for counting the number of shingles shared between each document later.
- The blacklist – a list of documents to be removed from the corpus because they shared too many shingles with another longer document.

tecl actually performs the destructive removal of documents considered too similar by tender. The longer of the duplicates is kept. tecl takes one or several XML files and blacklists as input (for example when tender and texrex were run in multiple processes), and merges them to create the final corpus. The output is one XML file, merged from the input files.

HyDRA and rofl

In addition to the main trio – texrex, tender and tecl – there are two more applications included in the texrex package that post-process the finished output: HyDRA and rofl. While there is no documentation for why these additional tools are included or needed, they do provide fixes to the corpus that can seem reasonable to perform.

```
<doc url="http://21days.withemp.com/" id="7cb74706fbf96e777fdcc7341e28f0281da8" ip="209.143.158.10"
sourcecharset="UTF-8" sourcedoctype="XHTML" bdc="c" bdv="3.16311" nbc="4472" nbcprop="0.866499" nbd="4"
nbdprop="0.222222" avgbpc="0.245966" avgbpd="0.802896" host="21days.withemp.com" tld="com"
language="unknown" country="US" date="Wed, 24 Aug 2016 17:45:56 GMT" last-modified="unknown" region="TX"
city="Houston" author="unknown">
<meta name="arcfile" content="./CC-MAIN-20160823195812-00261-ip-10-153-172-175.ec2.internal.warc" />
<meta name="arcoffset" content="2044194" />
<meta name="arclength" content="28656" />
<meta name="tarcfile" content="ENCorp_00_0000000001_2016-12-19_18-58-51.tarc.gz" />
<meta name="tarcheaderoffset" content="10" />
<meta name="tarcheaderlength" content="77" />
<meta name="tarcbodyoffset" content="88" />
<meta name="tarcbodylength" content="29736" />
<title>
Wisdom for Authorpreneurs, and Aspiring Authors and Entrepreneurs, from the Original English Word Nerd
Rebecca Woodhead
</title>
<keywords>
join-rebecca.com rebecca woodhead authorpreneur marketer internet marketing wealth aspiring author social
media influence
</keywords>
<div idx="1" bpc="k" bpv="1">
What isElite Marketing Pro? Magnetic Sponsoring NowOffers 100% Commission, and more ...
</div>
<div idx="3" bpc="a" bpv="-0.116593">
My name is Rebecca Woodhead and I did NOT get into this to make money, but that&apos;s what happened
anyhow. On this page, you&apos;ll see the story of my first three weeks, when I went from someone who
wanted to help her clients with cashflow to being the top recruiter in the company, and beating people
like Jonathan Budd and Ray Higdon on the Magnetic Sponsoring leaderboards! Here&apos;s the story of those
21 days ...
</div>
<div idx="4" bpc="j" bpv="0.882248">
THE FIRST STEP: I made the video below on the day I joined.
</div>
<div idx="5" bpc="k" bpv="1">
Scroll down to see my results after the $10 trial and after 3 weeks...
</div>
<div idx="6" bpc="f" bpv="0.509163">
Only promote companies and products that you feel ethical about promoting. This company is a stable, long
term business, not a flash in the pan. The products are proven to be some of the best in online marketing.
This stuff is gold! Click the first banner to see the quality of the squeeze pages and sales pages you get
as part of the system to promote them. No work for you. Ready made and ready to go! Forget 100%
commissions on businesses with no products, or shoddy products. Imagine getting 100% commission on THIS
quality of products. You already KNOW these sell. They are BESTSELLING products. You could take all the
profit. Watch the videos below to see what that could mean to your wallet.
</div>
<div idx="9" bpc="k" bpv="1">
ResultsAfter the 10 Day $10 Trial ...
```

Figure 4.2: An example extract of a document produced by texrex

HyDRA or **Hy**phenation **D**etection and **R**emoval **A**pplication, does what the name entails: it removes hard hyphenation, which is hyphenation inserted in words to signal a line break, often used in print where there is no room in a line for a word. The corpus has removed the context for which the line break was necessary and will therefore need to detect the occurrences of line break hyphenation that is no longer necessary.

hyphe– nation → hyphenation

An example of hard hyphenation being removed

Why these hard hyphenations occur in web sites is not immediately intuitive, as it does not seem usual for text to be edited to fit in a set size on the web. Rather the opposite, the web site dynamically adjusting the size of its containers for the text seems to be the norm. Transcripts from printed text may be one source of hard hyphenation on the web, but in these cases it can be discussed whether one would want to fix the occurences, or let it stay true to the original content.

The program takes a list of unigrams with frequencies that are to be deemed valid words as input, and removes hyphenation if the removal of it produces a word from the unigram list. In addition to the unigram list, the corpus is also needed as input. The output is the corpus, but with the hyphenation removed. The input corpus does not need to be in the output format of texrex, as long as it is text.

rofl In addition to HyDRA, there is one more post-processing tool, rofl – or run-on fixer lab. This is "a tool to fix run-together sentences based on word lists" (Schäfer, 2014, p. 2). Sentences like "This sentence ends in punctuation.And this sentence has no white-space after said punctuation" will be split up in two sentences.

Unlike HyDRA, rofl does not take unigrams with frequencies as input, but rather a list of valid words, in addition to the corpus. As there are several examples where no whitespace around punctuation is intended, rofl also supports specifying lists of prefixes and suffixes to ignore if found right before or right after punctuation. This is needed to avoid splitting urls ("nrk.no" would become "nrk. no" without 'no' listed in the suffix list). Additionally, one can make rofl ignore a line completely if it matches a user-specified regex string. This is useful to skip lines of meta data.

Like HyDRA, it is not obvious why such a tool is considered to be needed. There might be lacking white-space between sentences where non-text markup worked as a sentence splitter before said markup was removed. If this is the case, then white-space will need to be inserted, and a tool like rofl makes sense. But in other cases, it is not obvious whether editing the original content is wanted, even though we might disagree with the way the content was written. See the discussion in section 4.2.4.

4.2 texrex with the Common Crawl on a SLURM cluster

The processing of the Common Crawl using texrex was performed on the Abel cluster, located at the University of Oslo and open for scientific use to mainly researchers, as well as master and PhD students. The cluster provides more than 650 nodes, with a typical node having 16 cores running at 2.6 GHz and up to 64GB of memory. The management system for the cluster is the SLURM cluster management system[3], which Abel has configured so a single user is allowed to queue up to 400 jobs at the same time. The amount of jobs actually running at the same time varies, depending on the load on the system as well as the priority and size of the jobs, with smaller jobs having higher priority.

4.2.1 Running the texrex tool

With the Common Crawl downloaded and texrex installed, the question is how one would like to configure the process, and how it should be run in an efficient way.

Configuring texrex

texrex can be configured by specifying over a hundred different parameters. The parameters one can specify are related to either the texrex process itself (how many threads should the process use, for instance), the output (i.e. what should the prefix of the output files' names be?) or configuring the task itself (for instance: Should a subtask, like the boilerplate detection be enabled? If so, what neural network should it use?).

It is the specification of these parameters that make up almost every decision about the design of the corpus is made. As almost every subtask of web corpus construction have some decisions that can be debated, and will depend on what one wishes to accomplish, this part of the process is neccessary to dwell on. While I will not present every possible parameter here, as the texrex manual covers them thouroughly, I will discuss the choices I consider most important.

The subtasks were all set to enabled, except for the "TextAssessmentMulti", which enables the text quality assessment to work with several languages – as many as one includes a language profile for. For this corpus, I want to limit it to English. This means that the following were enabled and configured:

[3]https://slurm.schedmd.com/

- **Duplication detection**

  The duplication detection includes removing true duplicates and producing shingles for the near-duplication removal in the next tool in the tool chain – *tender*. There are a couple of key decisions to make for this step:

  - The size of each document's fingerprint: Set to 64 bytes.
  - Shingles are n-grams, but what should *n* be? Broder (1997) use 10, while the manual and Schäfer and Bildhauer (2012) use 5. I went with 5, as well, as the paper's results using 5 were good.

- **Size filtering**, the removal of too large or too small documents, is done twice, once while reading files (before any HTML stripping) and once after the HTML is stripped. This is the first step that performs removal of documents, before the boilerplate detector and duplicate remover performs additional filtering later.

  Recall the amount of documents removed per step from table 3.9, where the size thresholds ended up removing a lot of the crawled documents. Because of this relatively high amount of removed documents due to size, it is natural to question if the thresholds here and after boilerplate classification are set too aggressively.

  The purpose of the lower size thresholds are to remove documents that is assumed to end up with little to no text after cleaning. The upper thresholds are set for the documents that are very large, and that will be expensive for the more costly algorithms to process, like the boilerplate and duplication detection. The texrex manual (Schäfer, 2014) recommends discarding documents above 256 or 512 kilobytes.

  I followed the recommendations of the manual and from Schäfer and Bildhauer (2012), and set:

  - Min. size of non-cleaned document: 2kb
  - Max. size of non-cleaned document: 512kb
  - Min. number of texrex-separated paragraphs (after HTML stripping): 2
  - Minimum character count per document (after HTML stripping): 1000

- **UTF-8 validation** is done with IBM's ICU, where the documents' encoding is identified and converted to UTF-8. If the encoding is invalid, the document is removed.

- **Boilerplate detection** has many configurable parameters, from which neural network to use (recall section 2.3.2) to where the boundary between boilerplate and non-boilerplate should go. My settings were:

- ◦ The neural net to use for classification: texrex comes with some neural nets that are already trained. The one chosen was the one trained for English-, as per the results of Schäfer (2016a), which gave higher precision/recall/F1-scores for a neural net trained specifically for English.
- ◦ The threshold for when to count a paragraph as boilerplate: 0.5. Thresholds between 0.3 and 0.7 all provide decent accuracy and precision/recall. The 'optimal' cut-off point (where $precision = recall = F_1$) varied for different data sets (a data set per language), between 0.49 and 0.62. But as he concludes:

  > "The accuracy is virtually unchanged when moving from the balanced thresholds to the default threshold of 0.5" (Schäfer, 2016a, p. 8)

- ◦ Minimum amount of non-boilerplate paragraphs in a document to keep it: 1
- ◦ Minimum relative amount of non-boilerplate paragraphs to keep document: 0.1
- ◦ Minimum amount of non-boilerplate characters to keep document[4]: 500
- ◦ Minimum relative amount non-boilerplate characters to keep document: 0.25
- ◦ For the four thresholds above, I went with the default metrics, as they seem like a good compromise between keeping as much data as possible at the same time as removing documents that are almost completely composed of boilerplate.

- **Text quality assessment and language detection**, also called "Text Assessment" or "Connected text identification" requires tokenisation for the token counts, so I include the parameters for tokenisation as well.

  The tokeniser of 2012's texrex (Schäfer and Bildhauer, 2012) worked by considering anything between two non-letters in the ISO-8859 encoding as tokens. As texrex is now using UTF-8 as its encoding, a similar but slightly different tokenisation method is surely used. However, I have not found any updated information about this.

  - ◦ The threshold for when to count a paragraph as boilerplate when counting tokens for the unigram list: 55
  - ◦ Minimum char length of div to count the tokens: 100

[4]Note that the size filter right after HTML stripping removed documents smaller than 1000 characters. This setting also takes the amount of non-boilerplate characters into account.

- Language profile used: The English profile provided by texrex
- Recall the Badness value described in chapter 2.3.2. I went with Schäfer et al. (2013)'s recommendation of a threshold at 35.

Running texrex

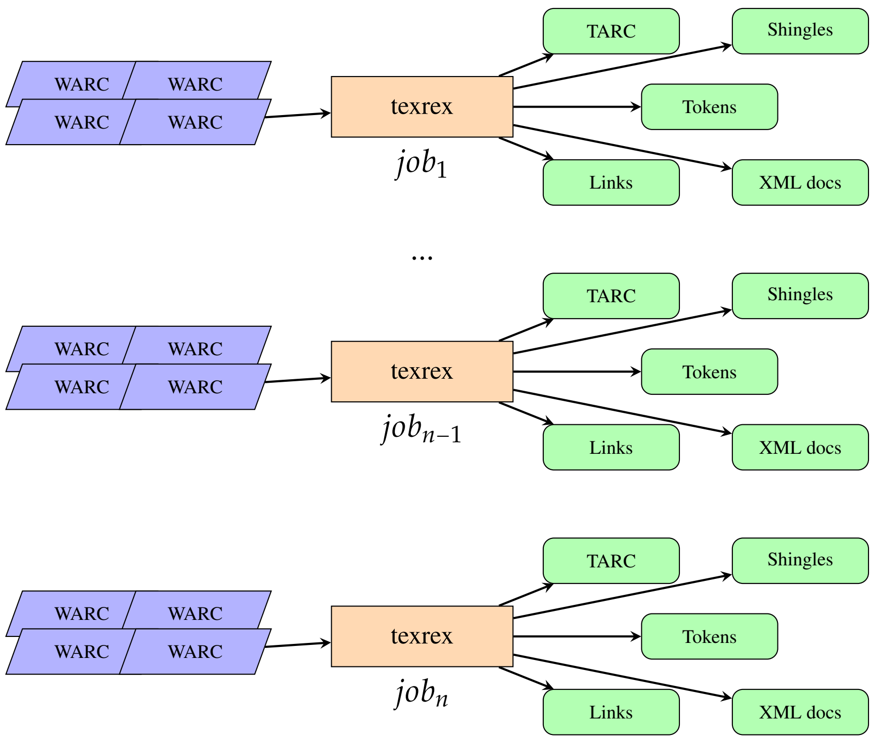


Figure 4.3: How texrex was run – each texrex process got 3–4 WARC files as input

The decision of how many files per job (and as an extension, how many jobs to run) was taken for me, as texrex would constantly freeze after some time when run with more data than 3-4 GB of compressed Common Crawl data. This made me decide that about 3 files (recall that each file from the Common Crawl is about 1GB) per job made the most sense. With 29,800 files, this ended up being around 9930 jobs, more than the queue limit of 400 of the SLURM system. A script continously queueing new jobs as others on my list finished helped complete the entire process in about 8 days, where each individual job ran between 1 and 4 hours each.

Each job used 8 threads and were allocated 8 GB of RAM – however, the jobs used around 3 to 7, depending on input size. See table 4.1 for an overview of how the jobs were run on the cluster. As stated earlier, the amount of time used

process	# jobs	thr./job	alloc. mem. (GB)	hours/job	hours total
texrex	10k	8	8	1 – 4	200
tender	995	1	16	0.5 - 1 (72)	100
tecl	498	1	12	3	48
HyDRA	71	1	2	10	12
rofl	71	1	2	16	20

Table 4.1: A breakdown of how I ended up setting up the jobs for the different processes

to complete the jobs depends heavily on the cluster's load, so one's mileage may vary, even on similar systems.

About 150 jobs had either timed out or crashed, with no output as the result. Rerunning these jobs saw that number decline to about 30 jobs that would not finish successfully. Inspection revealed a similar problem to the one seen when trying to run texrex with too much data, and so I set up new jobs with this data, leaving one file per job. Every job proceeded to complete successfully.

4.2.2 A tender deduplication

The next tool in the chain, tender, can take in several shingle files output by each texrex job, sort them and produce a blacklist of duplicate documents that should be removed.

It is possible to configure the tool, but unlike the extensive configuration options of texrex, tender is configured using simple command line options. The most important one is likely the *--limit* option, the threshold for how many shingles that need to overlap between documents to consider them near-duplicates. The default is 5, which is also used in the paper, and is what I used as well.

As tender needs to compare each document's shingles with each other, there needs to be one single superprocess that in the end takes all the shingles as its input. Luckily, tender also supports taking blacklists that have already been produced by other tender processes as input, skipping the documents that already occur on the blacklist.

The first try Initially, I tried running tender concurrently by splitting the task in several 'layers' of jobs, where the first layer would consist of 995 jobs having shingle files from 10 texrex processes per tender job, going from about 9950 shingle files to 995 blacklists. I then planned to continue to merge groups of output from processes from the previous layer to keep running processes concurrently

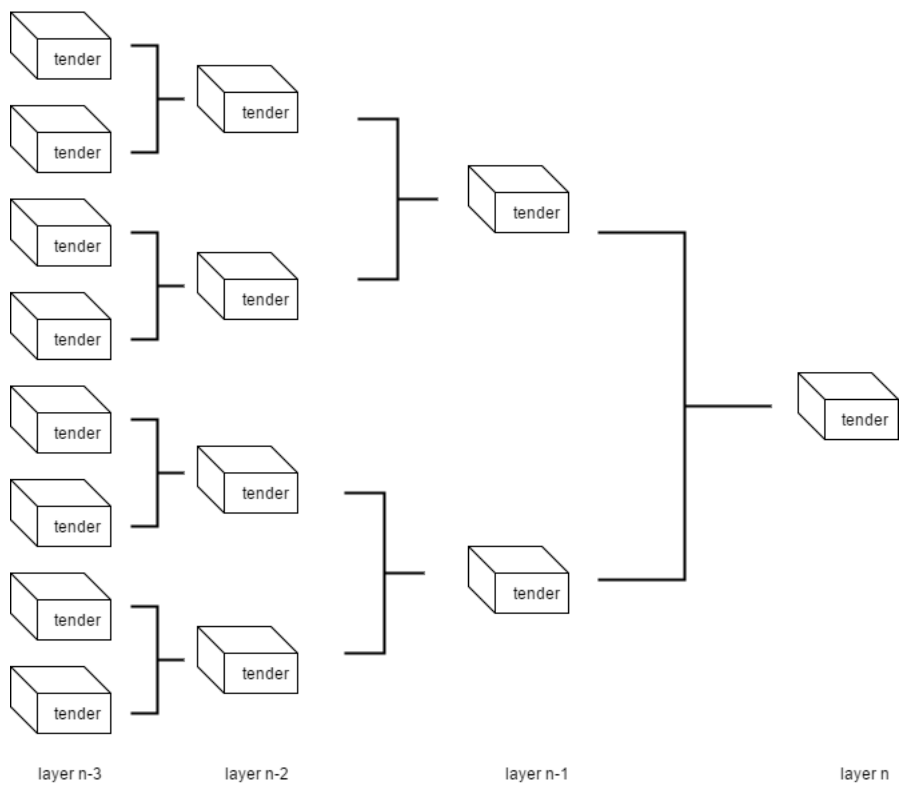


Figure 4.4: The abandoned idea of merging tender processes until ending up with one

– see figure 4.2.2. The premise was the hope that the blacklists I ended up with would comprise so many documents that the last process would not need to process that many documents.

The approach had two problems which caused me to abandon it:

1. I did not realise that the blacklists output from a tender process did not include the documents from the blacklist that was input. Therefore, as I progressed through the 'layers', I ended up reprocessing documents needlessly because I assumed they would be included in the blacklist for the next layer.
2. The premise that the later layers would need to process far fewer documents was false: Because a small percentage of the total amount of documents are near-duplicates, a correspondingly low amount of documents are listed in the blacklists, making the later layers skip rather few documents.

The successful setup The setup that proved to work was to split the shingle files from texrex similarly as my initial try: 995 tender jobs having shingle files from 10 texrex processes per job. However, instead of incrementally merging layers of processes, I fed the blacklists of the first 995 tender processes to one single tender

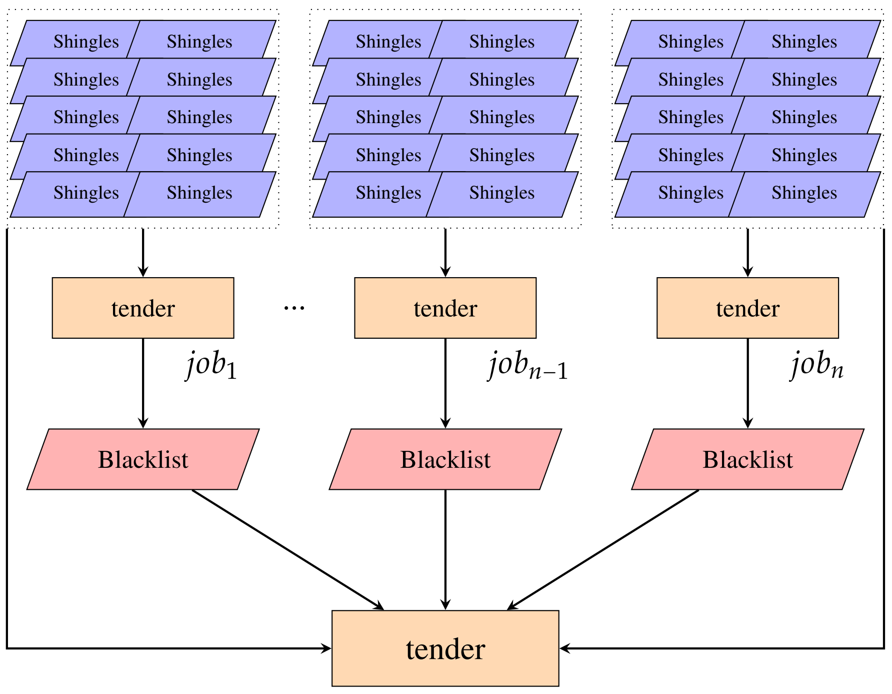


Figure 4.5: The tender setup: tender jobs produce blacklists, which goes into a final tender together with the shingles

superprocess that processed all the shingle files. Each of the 995 jobs from the first ‘layer’ finished in between 20 minutes to 1.5 hours, with queueing time making the whole layer finish in about 15 hours. The final superprocess completed in approximately 3 days.

4.2.3 tecl-ing the near-duplicated documents

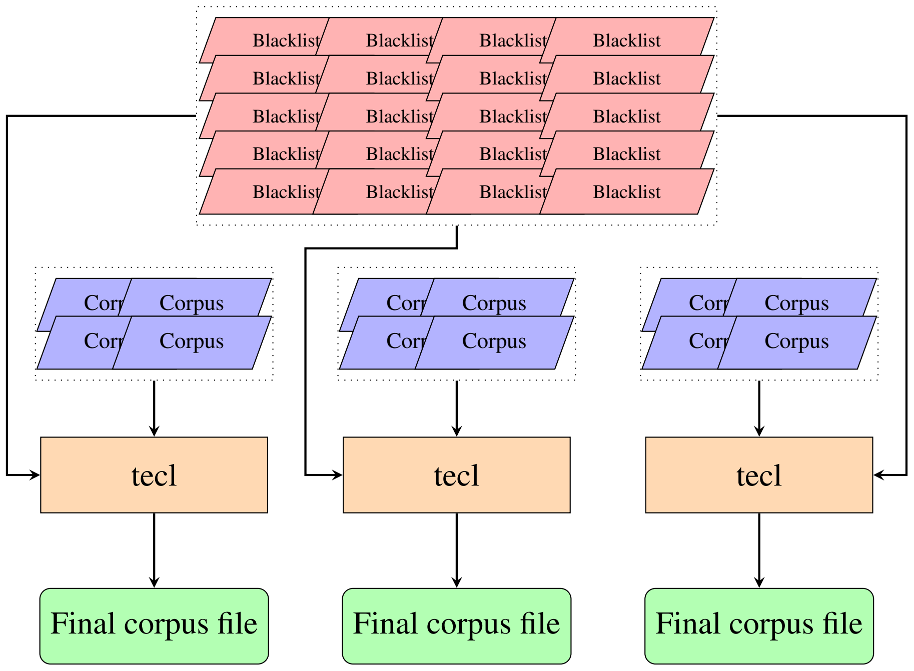


Figure 4.6: The tecl setup – corpus files and blacklists go into tecl and result in merged corpus files

tecl takes tender's blacklists and texrex' corpus files and outputs new corpus files with the near-duplicates listed in the blacklists removed. Like tender, tecl only supports one thread. However, splitting the input over several independent processes is easier with tecl. I solved this by splitting the 9950 corpus files from texrex so that each tecl process processed 20 corpus files each, making the amount of tecl processes 498. These used about 3 hours each, and with queueing time every job was finished after two days.

The final output is 422 GB of compressed deduplicated corpus files, stored as one file per tecl process, namely 498.

4.2.4 HyDRA and rofl

The corpus is at this point mainly finished, but the post-processing tools HyDRA and rofl are still left. I had doubts whether these steps were neccessary or even wanted, as mentioned in section 4.1.2, and decided to complete them, inspect the result and decide which of the corpora I wish to continue with – the processed or the non-processed.

I queued 71 jobs on the cluster, each job running HyDRA on 7 files each. For each HyDRA job, a job running rofl were queued as a dependency of the HyDRA job, taking the files from HyDRA and processing them with rofl. Each individual HyDRA job finished in about 10 hours, and the rofl jobs used 16 hours each. All 142 jobs finished in about 32 hours.

There were about 150 million run-on sentences that were split, around 1.5% of the total amount of sentences, a surprisingly large amount. Inspecting 100 randomly selected changes reveals that a lot of the fixed sentences originate from forums, chats, comment sections, and similar user-generated content, where a user has omitted a white-space. Whether or not a corpus creator should be correcting content creators' grammar can be debated, but my opinion is that there is value in content being true to the original source.

Assessment	Count
User-generated content	42
Incorrect	20
Correct	38
Total	100

Table 4.2: Randomly selected rofl changes manually assessed to either be user-generated, correctly changed or incorrectly changed

There are also occurences of parts of URLs that the prefix list cannot pick up, like "docs.microsoft.com" becoming "docs. microsoft.com". Other examples of incorrect changes include source code examples using punctuation, email subjects like "Re:<subject>", file path extensions, usernames with punctuation in them, and so on.

For HyDRA, there were about 8 million occurrences of hyphenations that were changed. This is about 0.006% of the total amount of tokens. Rough inspection of the removed hyphenation did not reveal any mistakes – the changes seemed reasonable.

Although some corpus users might be interested in post-processing steps such as these being done, others might be interested in the truer-to-source content. As these changes are destructive and cannot be undone, I will be continuing without these post-processing steps applied.

4.2.5 xml2txt

The texrex tool chain never performs destructive operations on the documents it has kept, to retain as much original context as possible. This means that even though texrex has considered paragraphs to be boilerplate, the paragraphs are not actually deleted. To extract the non-boilerplate text, it is therefore necessary to process the XML files and retrieve the text where the boilerplate classification score for a div is below the threshold.

This was solved by developing a script extracting the relevant information from the different XML elements, before collecting the raw text from the paragraphs below the threshold. Only the paragraphs with a boilerplate score below a value input by the user of the script (0.5 here) is kept in the text version.

Parsing the XML is also possible, but a robust parser is required, as there are a few (around 100 in total for this corpus) mistakes, where it seems like texrex has overlapped two documents, even sometimes beginning a new document inside an XML tag of the previous document.

The result of the text extraction is about 115 billion white-space separated tokens, split in files containing 10,000 documents per file – bringing the total amount to 8688 files at an average size of 74 MB. Each document is separated by a form-feed character (an ASCII control character, represented as '\f' in C and other programming languages), and each sentence is separated by a new-line character ('\n'). To keep the link between the extracted non-boilerplate text and the original document, the script generates .meta-files where each line represents a document, with the file name of the origin XML file and the byte offset to find said document listed.

To see how much boilerplate that is still kept, I ran xml2txt once more, only extracting boilerplate instead of extracting non-boilerplate. This was done by keeping the paragraphs above a boilerplate score of 0.5. The result was 45 billion white-space separated tokens of boilerplate text – a noticeably large amount.

4.2.6 Tokenisation and conversion to CoNLL

While the corpus now exists in both an XML and raw text-format, tokenising it using a more advanced tokeniser than a white-space tokeniser is a logical next step. I used Stanford's CoreNLP (Manning et al., 2014) for tokenising. The 8688 text files were split to 49 jobs of CoreNLP tasks, configured to only tokenise and sentence split. CoreNLP can do other tasks as well, like part-of-speech–tagging, lemmatisation, Named-Entity Recognition, etc. However, these tasks are expensive to perform, and was not prioritised.

The output is in the CoNLL format, originally created for the yearly shared task at the Computational Natural Language Learning conference. Slight changes on the structure of the format occur from year to year, but the CoreNLP website writes that their CoNLL format does not correspond to any specific existing formats[5]. Each line represents a token, and each line has tab-separated information about said token. Normally, this is information like the lemma of the token, the Part-of-

[5] https://stanfordnlp.github.io/CoreNLP/cmdline.html

Representation	# Files	Total size (GB)	Avg. file size
XML	498	1376	847 MB
Raw text	8688	645	74 MB
CoNLL	8688	2300	264 MB

Table 4.3: Structure of the corpus files. The raw text also come with as many meta files using about 5GB of space.

Speech tag or similar annotation. As this corpus is only tokenised and sentence split, the index and token is the only information per line.

Many NLP tools can read the CoNLL format directly – which makes it a popular format, and the latest CoNLL-U format is a de-facto standard within the field.

The 49 jobs each ran between 1 and 3 hours. Although each job got the same amount of files, and each file contains the same amount of documents, each document is not the same size. Therefore, some files are a lot larger than others, and this was evident when running CoreNLP. The memory requirements were large, and several of the jobs had to be re-queued because they ran out of memory. For the jobs requiring the most, I had to allocate 24 GB of memory.

4.3 Presenting enC^3: the English CommonCrawl Corpus

As the size of the corpus is large, seemingly trivial steps such as extracting text from XML, tokenisation or sentence segmentation can take a long time. There are differences in the needs of corpus users, as well as differences between what kinds of resources users have access to – everyone does not have access to a cluster, for instance. Because of this, the corpus is made available in three ways:

- XML
- Raw text + linker files to the XML
- CoNLL

4.3.1 XML

The XML representation is the primary format of the corpus, where all original meta data is available and where no text has been removed from the documents, even though the classifier has deemed it boilerplate. This representation is for the users that might want whole documents in their original context, that wishes to extract text based on another boilerplate threshold, that wishes to perform some additional post-processing, or is in need of some type of meta data.

An example document can be seen in figure 4.2. The documents are within <doc> and </doc> tags, while <div> and </div> surrounds each paragraph. The attributes inside the paragraph tags describe whether or not the paragraph is considered boilerplate.

4.3.2 Text

The text representation contains no meta data and text deemed boilerplate is not included. The text is extracted as described in section 4.2.5. A corpus user who is interested in extracted, non-boilerplate web text, but perhaps not interested in doing XML parsing or in the technicalities of how to consider a paragraph boilerplate will probably want to use this format.

For each file, there is an accompanying meta file, which says where the documents can be found in the original XML files, for those who wishes to retrieve the information that was removed in the text representation.

4.3.3 CoNLL

Additionally, the corpus is available in the CoNLL format, as output by Stanford's CoreNLP. These files are tokenised and sentence segmented based on the text representation, and is ready to be input directly in many NLP tools using the CoNLL format as its standard. This is the easiest format to use for those who wish to use the corpus further, and that agree with the decisions made in the steps between the XML representation and this one.

4.3.4 enC^3

As of this writing, I am not aware of any corpora, cited or otherwise, that are larger in terms of token count than enC^3. However, the document count is not as large as *enTenTen*, which means that there are more tokens per document in enC^3. Some variations between data sets must be expected, but a difference this large might indicate a difference in what types of documents were kept, and might be due to how texrex removes documents with too little non-boilerplate content.

enC^3 contains:

- 86 million documents
- 6 billion sentences

Corpus	docs	sent.	tokens	t/d	t/s	s/d
enC³	86	**6,060**	**135,159**	1571	22	70
enClueWeb	**139**	—	81,991	590	—	—
enTenTen	—	—	23,000	—	—	—
DECOW	17	807	20,495	1195	25	47
GigaWord	9	—	4,033	408	—	—
Wikipedia14	5	—	1,715	375	—	—
ukWaC	3	—	1,914	712	—	—

Table 4.4: Some corpus statistics from selected large corpora. t: tokens, d: documents, s: sentences. All absolute numbers are in millions. Relative numbers (e.g. tokens per document) are as they are.

- 135 billion tokens

In table 4.4, enC³ is compared with the other large corpora I am aware of. Some of them, like *enClueWeb* (Pomikálek et al., 2012) and *enTenTen* (Jakubiček, Kilgarriff, Kovař, Rychly, and Suchomel, 2013) use either the SpiderLing toolkit (Suchomel and Pomikálek, 2012), or components from it – like *jusText* for boilerplate detection and *onion* for duplication detection. ukWaC (Ferraresi, Zanchetta, Baroni, and Bernardini, 2008) is the largest WaCky initiative corpus, while DECOW is the largest of the COW corpora, also made with texrex (Schäfer, 2015).

In addition to the web corpora, traditional corpora like the *English GigaWord Fifth Edition* (Parker, Graff, Kong, Chen, and Maeda, 2011), gathered from different news sources, and extracted text from a 2014 Wikipedia dump[6] were compared as well.

Not surprisingly, the web corpora are much larger than the traditional corpora: enC³ is about 77 times as large as GigaWord in terms of token count.

[6]http://linguatools.org/tools/corpora/wikipedia-monolingual-corpora/

Chapter 5

Evaluating the English Common Crawl Corpus

Kilgarriff et al. (2014, p. 1) describe how cooks preparing food depend as much on the quality of their ingredients as their own skill, and draw a parallel to language technology:

> Be aware of your sources, you want your produce to be from a farmer who cares about quality. So it is with language technology. Those writing the applications are the cooks, those preparing corpora, the farmers. The applications are crucial to the success of the enterprise. But – and this becomes inescapable as more and more methods are based on learning from data – so too is the quality of the data they are based on.

The paper goes on to describe how there has been a tradition for at least 20 years for evaluating language technology systems. For these evaluations, the same data has typically been used, with the purpose of leveling the playing field and attributing the prospective better results to the improved system. Specifically, the "same data" that is used are often corpora, and for the purpose of comparability to other systems' results, the same corpora are used for all evaluations.

As was argued with the "chefs and ingredients" example above, making sure the system is performing well without regard for the quality of the input data is not sufficient to maximise the quality of the output. An example of this, as mentioned in chapter 2, is Banko and Brill (2001)'s experiments. They showed that for some selected tasks, the performance boost of increasing the size of the input was larger than fine-tuning the machine-learning algorithms.

Conversely, Versley and Panchenko (2012) quote several experiments where large web corpora perform worse than much smaller manually constructed corpora. Similarly, the experiments of Biemann et al. (2013) show that the performance of their chosen task, matching corpus collocations with a combinatorial dictionary, increase with the size of the web corpus, but gives better results with the smaller, manually constructed British National Corpus regardless.

The differing results might be due to corpus quality, but some task results can perhaps be attributed to the tasks being biased to some text type or domain. Tasks testing for words typically present in news might do better on collections of news text, while other tasks requiring high coverage of proper nouns might perform well with encyclopedic text. While many desire web corpora that do well with a multitude of tasks, including tasks like the previous examples, it is important to note this difference in types and domain, and that some tasks might see their results affected.

As more corpora become available, corpus users might wonder which corpus to choose for their task, and corpus creators need to know whether or not they have done a good job.

Hence, there is a need for some way of evaluating corpora, which could either be done by inspection of the corpus or by using it for downstream tasks, and comparing the results to other corpora used in the same task.

5.1 Inspection vs. downstream task experiments

Language technology systems are often evaluated in an *intrinsic* or an *extrinsic* perspective.

5.1.1 Intrinsic evaluation

Intrinsic evaluation is when the quality of a system is assessed by direct analysis of the system's characteristics, and how they relate to some given criteria. Often, the hope is that good results in an intrinsic evaluation will be telling of a system's quality and of its aptness for further use in downstream tasks (however, this assumption might not always be true – see discussion in section 5.3.3).

Similarly, inspection of corpora and collection of corpus statistics can make sure that some criteria hold true – like the absence of duplicated content, or avoiding host bias. To illustrate, Biemann et al. (2013) use several corpus statistics – such as the most frequent words, the length distribution of sentences, etc. – to identify problems, or the lack of them, such as for instance many malformed sentences.

5.1.2 Extrinsic evaluation

Unlike with intrinsic evaluation, one who performs extrinsic evaluation does not assess or inspect the system directly. Rather, the system is assessed by the use of its output as input in another downstream task. The results of this downstream task is then indicative of the quality of the original system, often by comparing the task results with the results of the same task using other input. However, extrinsic evaluation using one given task may not represent the quality of the system for use with a different task. Thus, good results in one task do not necessarily imply good results in another.

As stated earlier, corpora are used as data for many machine learning based language technology tasks. These tasks are candidates for evaluating corpora extrinsically, and some of them have been used for that purpose: Kilgarriff et al. (2014) describe a method for extrinsic evaluation of corpora using a collocation dictionary task, while Biemann et al. (2013) use a combinatorial dictionary collocation task and a verb-particle combination task.

5.1.3 How to evaluate the English Common Crawl Corpus

Evaluating how well the sub-tasks solved during corpus construction performed can be useful information. Did the boilerplate removal remove sufficient boilerplate? How much duplication is left? Additionally, corpus statistics to identify other problems with the content can also be helpful.

However, obtaining every convenient metric to get a fuller understanding of the corpus is, sadly, too time-consuming. The scale of this project requires choosing one evaluation method. The primary goal of that evaluation is to be able to get a sense of how enC^3 fares compared to other corpora in a real-world application, and to get an indication of whether constructing corpora from the web is worth the effort. Hence, choosing a downstream task and comparing the results using different corpora will be the chosen method of evaluation.

While there probably never will exist a task where good results for that task correspond with good results for all tasks, some probably lend themselves better to evaluation purposes than others. Tasks where the result greatly depend on the input data and where a good result can be easily discerned from a poor one are preferred.

Presently, there is no standard downstream task for corpus evaluation – different constructed corpora are evaluated using different methods, and no one method seems more usual than others. For that reason, the chosen task will not be selected based on what others have done before. Instead, in addition to the points made in the previous paragraph, the task will be selected based on how useful the results will be and to keep the complexity of the task as simple as possible. Low complexity tasks are often less time-consuming, and have fewer factors that contribute to the result, making it easier to attribute it to the corpus.

One such task that shows disparities between the results of using ‘poor’ corpora and ‘good’ corpora, as well as being of large interest to the Natural Language Processing community is the training and evaluation of so-called *word embeddings*, or dense word vectors. This will be the chosen task for the evaluation of enC³. Before presenting and discussing the different word embedding models and the chosen benchmark tests, I will provide some background on distributional semantic representations.

5.2 Embedded word vectors in low-dimensional spaces

Distributional semantic representations in the form of word vectors have seen an increase in popularity in the last couple of years. This new-found interest has been particularly apparent when it comes to representing words as *word embeddings* – or dense vectors embedded in low-dimensional spaces.

Representing words as vectors has a long tradition in NLP, and these traditional representations are often referred to as ‘count-based’ models, contrasted with the new generation of ‘predict-based’ models. This distinction is not clear-cut, however, and Fares et al. (2017) instead suggest separating explicit representations, “where each dimension of a high-dimensional and sparse vector directly corresponds to a contextual feature” and the continuous low-dimensional word-embeddings.

5.2.1 Explicit word representations

The purpose of constructing vectors representing words is to create a spatial representation of word meaning. In this geometric representation, words that are close together in the vector space will often be semantically similar as well. The idea is that words that are used in a similar way often have similar meaning as well, and thus will co-occur with many of the same words (Harris, 1954; Firth, 1957).

Traditionally, word vector representations have as many dimensions as there are words in the vocabulary, where each word represents one dimension. The construction of the word vector is done by counting how many times the given word co-occurs with the other words in a specific context (within the same sentence, document, etc). To illustrate, one can imagine the following constructed example:

s_1: I play guitar in the band.
s_2: The band needs someone to play drums.
s_3: Don't make me play football.
s_4: Sport and music go hand in hand.

The vocabulary of these sentences contains 19 word types, which makes for 19-dimensional vectors. In practice, a realistic model will have a lot more sentences, numerous word types, and thus thousands of dimensions. It is impossible to visualise such high-dimensional spaces, but it is possible to get a sense of what such representations look like by considering two dimensions, and a few selected words:

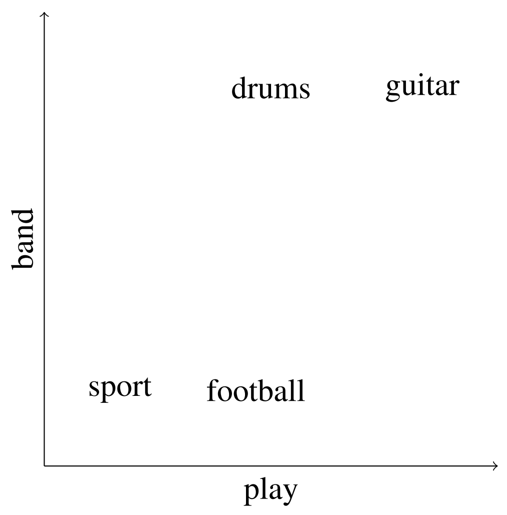


Figure 5.1: The vector representation of selected words from s_1 to s_4 with 'band' and 'play' as axes

In this perhaps contrived example, the idea is to see that the words 'drums' and 'guitar' are close together, and as a group far away from the words 'sport' and 'football'. This also corresponds with an intuitive understanding of the words' semantic relations: guitar and drums are oftentimes played in bands, unlike a football.

The explicit models are in reality more advanced than this simple example, and do provide good results in a wide range of lexical semantic tasks. However, the newer approach of word embedding models performs better across the board and also has the advantage of scaling well to large data sets (Baroni, Dinu, and Kruszewski, 2014).

5.2.2 Word embeddings

The next generation of distributional semantic models uses various techniques to embed words as vectors into a lower dimensional space to create *word embeddings*

(Bengio, Ducharme, Vincent, and Jauvin, 2003; Collobert and Weston, 2008; Mikolov et al., 2013; Pennington et al., 2014; Bojanowski, Grave, Joulin, and Mikolov, 2016).

There are several models for training word embeddings, and for the purposes of evaluating enC³, one of the three most recent will be chosen: Word2Vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014) or fastText (Bojanowski et al., 2016). They are similar in that they all provide word vectors from an input corpus and perform well in the oft-used evaluation tasks (discussed in section 5.2.3).

Word2Vec

Mikolov et al. (2013) took earlier efforts further and can be said to have kicked off the current popularity of word embeddings with *Word2Vec*. Unlike the explicit representations and counting co-occurrences, Word2Vec is a prediction based neural network model that estimate the vectors by predicting words, and tweaking the weights of the input vector using back propagation based on the success of the prediction.

They provided two different architectures for the creation of the vectors – the skip-gram and continuous bag-of-words models. Both are shallow two-layer neural networks that use a continuous window of *n* words for their prediction. They are different in what they try to predict: the skip-gram model uses a single word to try to predict the *n* surrounding words. The CBOW model uses the *n* surrounding words to predict a single word. See figure 5.2.

GloVe

Interestingly, Pennington et al. (2014)'s ***Glo****bal* ***Ve****ctors* goes back to counting co-occurrences, criticising Word2Vec for using local context windows and not taking advantage of global corpus statistics. The paper identifies how 'certain aspects of meaning' can be extracted by using ratios of co-occurrence probabilities rather than the probabilities themselves (see table 5.1). They go on to train word vectors by minimising the difference between the dot product of the word vector and the logarithm of the words' probability of co-occurrence.

fastText

fastText (Bojanowski et al., 2016) is an extension on Word2Vec's Skip-gram model that also incorporates character n-grams in their model. That is, a word vector in the fastText model is the sum of the vectors of the character n-grams that make up the word. The advantage, according to the paper, is that rare words can be represented more accurately, as the character n-grams that make up the word

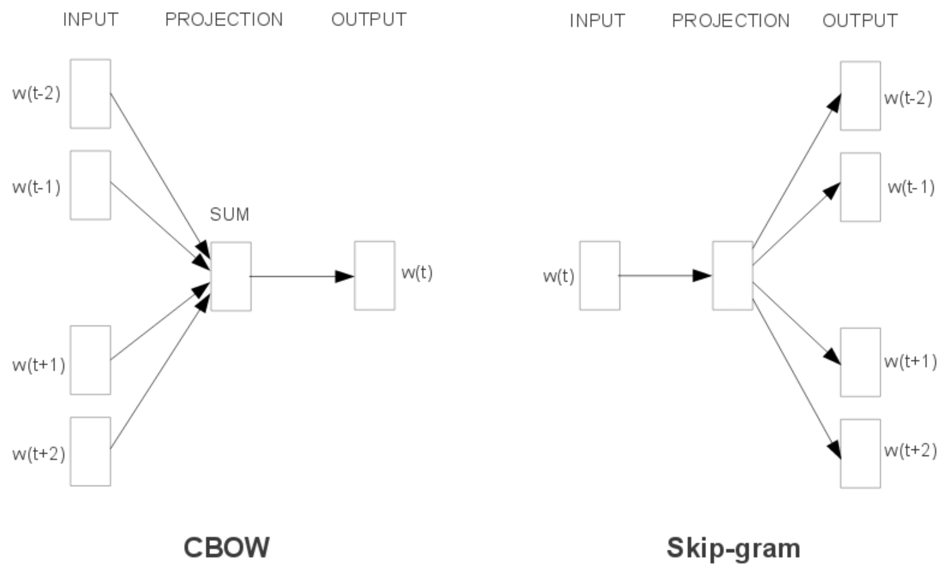


Figure 5.2: The difference between the skip-gram and the CBOW models. The CBOW model takes surrounding words as input, and tries to predict the word in the middle, while the skip-gram model takes a single word as input to predict the surrounding words.

Probability and ratio	k = *solid*	k = *gas*	k = *water*	k = *fashion*
P(k|ice)	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
P(k|steam)	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
P(k|ice)/P(k|steam)	8.9	8.5×10^{-2}	1.36	0.96

Table 5.1: Co-occurrence probabilities for target words *ice* and *steam*. Large probability ratios correlate well with properties specific to ice, while very small ratios correlate well with properties specific to steam (ice is solid, steam is a gas). Words that both target words relate to equally is about 1.

are seen not only in that word. This is especially important for morphologically rich languages like Finnish or Turkish, but there is increased performance even for 'richer-than-English' languages like German.

Choosing a word vector model

Although the models are different both in terms of implementation and method, the word vectors they provide perform quite similarly in evaluations, both in terms of absolute numbers, but also in the way they vary with input data. The goal of the word vector construction is not to get the 'best' word vectors possible, but instead to find a method of comparison for corpora. For this, all the methods would be suitable.

There are few points of reference in terms of how much time each method requires to train, and also how they scale to larger data sets. The few informal sources that do exist[1][2] made me lean towards GloVe, as it seems to require a lot of RAM, but the payoff is that it runs quicker. This fits with my situation, where I have access to a system with a lot of RAM, and where time is the largest constraint.

Additionally, the then–state-of-the-art evaluation results Pennington et al. (2014) reported were evaluated on word vectors trained on Common Crawl data, which is provided on the GloVe website[3] together with several other pre-trained vectors. This is an interesting point of comparison for evaluating enC³, and made me select GloVe as my word vector model.

5.2.3 Evaluation of word vectors

Unlike with the evaluation of corpora, there do exist oft-used, standard ways of evaluating word vectors. All the word embedding training architectures use these methods to compare themselves with each other. The methods are *word analogy tasks* and comparisons with human *word similarity* judgments. Both tasks are based on the word vectors' capability of encoding different kinds of similarity.

Word analogy tasks

Mikolov et al. (2013) presented the observation that word vectors encode more complex kinds of similarities than semantically similar words also being close in vector space (in terms of cosine similarity or euclidean distance) – like 'France' and 'Italy'. They designed a task that tests for other kinds of similarities, similarities that show relations and analogies between words.

The task is to solve the question "A is to B as C is to D", where D will be filled in using the word vectors, and A, B and C is given as input. For instance, the question "Oslo is to Norway as Athens is to ___" can be solved by vector algebra:

$$X = \text{vector}(\text{“Norway”}) - \text{vector}(\text{“Oslo”}) + \text{vector}(\text{“Athens”}) \tag{5.1}$$

Then the vector space is searched for the word closest to the vector X measured by cosine distance (disregarding the input question words in the search), and this word is used as the answer. The idea is that if the word embeddings are of sufficient quality, the intuitive semantic relationship matches the output. For the example above, "Greece" is expected as output.

[1]http://dsnotes.com/post/glove-enwiki/

[2]https://rare-technologies.com/fasttext-and-gensim-word-embeddings/

[3]https://nlp.stanford.edu/projects/glove/

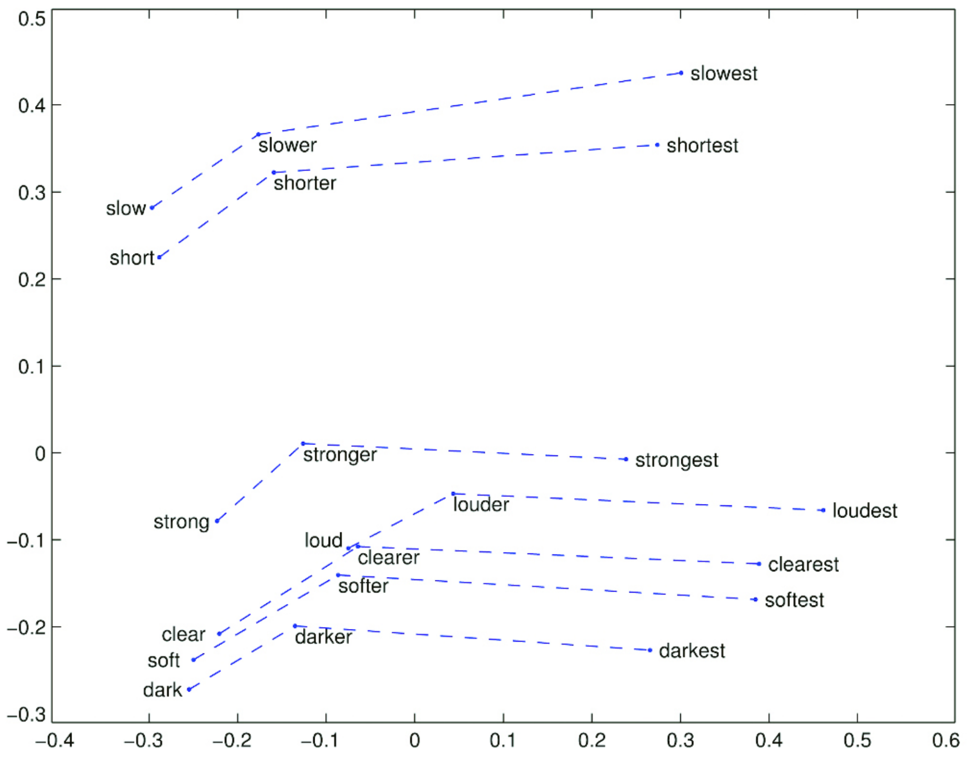


Figure 5.3: The vector differences encode the syntactic relationship between conjugated verbs

This method reveals that general relationships between words are encoded in the vector differences (see figure 5.3), and that this analogical similarity-task can find different types of relationships. Mikolov et al. (2013) created a dataset of 8869 semantic and 10675 syntactic questions. The semantic relationships that are tested include currency (Norway → kroner, USA → dollar) and female–male (brother → sister, grandson → granddaughter), while the syntactic word pairs test for relationships such as present participle (think → thinking, read → reading) and nationality adjectives (Denmark → Danish, Germany → German).

Word similarity

The analogy task tests for more complex vector space substructures, but the more traditional test for semantic similarity by looking for a word's nearest neighbour in vector space is also useful.

This is done by using lists of word pairs that have had their similarity assessed by human judges. There are differences in how these lists are compiled, how the similarity scores for a word pair are determined and what types of words that are included in the list. I will discuss the differences between some popular word similarity lists in section 5.3.2.

These word pair lists are then assessed using the word vectors, where a similarity score is found by calculating the cosine similarity between the words. In the end, there will be two lists sorted by the similarity score: One using the human judgments and one using the scores from the word vectors.

The word vectors are considered to perform well if the two lists have ranked the word pairs similarly. Specifically, Spearman's rank correlation coefficient (Spearman, 1904) is often used to find a number ranging between -1 (the lists' rankings are opposite of each other) and 1 (the lists' rankings are identical), and as close to 1 the score is, the better the word vectors are considered to perform.

5.3 Training and evaluating word embeddings with enC^3 data

Training word vectors on 135 billion tokens will be an expensive operation. Just like with the corpus construction, the Abel cluster will be used for the training.

5.3.1 Training word vectors using GloVe

Several parameters need to be configured for the model, and because the intention is to be able to compare the resulting word vectors to the word vectors of Pennington et al. (2014), the parameters are set equal to the ones the paper describes when training their example vectors (see table 5.2), except for the initial learning rate and the vocabulary size – see discussion below. In addition to the parameters, enC^3 is pre-processed in the same way the paper pre-processed their corpora: Tokenised using the Stanford tokeniser, lowercased and merged to one file.

GloVe training is split up in four processes:

- Creating the vocabulary
- Counting co-occurrences
- Shuffling the co-occurrences
- Training the word vectors

Building the vocabulary

Creating the vocabulary was a relatively inexpensive operation, and completed in about 15 hours, using one thread. Pennington et al. (2014) stated that for the smaller corpora, a vocabulary size of 400,000 was used, but for the larger

Param.	Value
Vocabulary size	2,000,000
Window size	10
Vector dimension	300
η	0.03
α	0.75
x_{max}	100
Iterations	100

Table 5.2: The parameters chosen for the co-occurrence counting and the training of the word vectors. The window size represents the number of context words to the left *and* to the right. η is the initial learning rate, α represents the exponent in the weighting function used in the cost function of the least squares problem and x_{max} represents the cutoff in the same weighting function.

Common Crawl based corpora, a vocabulary of "about 2,000,000 words" was used. The specific vocabulary sizes of their two large corpora did not match, however, and I speculate they really used a frequency threshold instead of a max vocabulary size (the real vocabulary sizes used for their two large corpora were 1.9 million and 2.1 million). I did not come to this realisation until after the vocabulary was built, however, and used a vocabulary size of 2,000,000 words.

Counting and shuffling co-occurrences

The co-occurrence counting program did not support multi-threading, and took about 35 hours. There is a customisable upper memory limit that, if reached, dumps the counted co-occurrences in memory to an overflow file and then continues. However, the program must merge these co-occurrence files later, and did not succeed with low memory requirements. The problem was solved when using 200 GB of memory. The resulting co-occurrence matrix file was 147 GB in size. The shuffling of the matrix then took a couple of hours.

Fitting the GloVe

Training the model was the most expensive step, requiring about 30 GB of memory and almost 8 days of run time with my setup of 32 threads, 100 iterations and a vector dimension of 300. A single iteration took around 1.9 hours. The resulting word vectors were 5.4 GB in size.

The paper originally used an η of 0.05, but this caused the process to output "-nan"

Process	Memory	Run-time	Threads
Vocab. building	5G	15h	1
Counting/Shuffling Co-occurrences	200G	40h	1
Training the model	30G	190h	32
Total	—	245h	—

Table 5.3: How the different processes were set up.

instead of the result of the cost function. There was advise on the GloVe forums to lower the initial training rate, and an η of 0.03 solved the problem[4].

The model generates two sets of word vectors, the vectors themselves (W) and the context word vectors ($\tilde{W}$). Pennington et al. (2014) write that when the co-occurrence matrix is symmetric (which it is when choosing a context both to the left and the right of the word), W and $\tilde{W}$ are equivalent, differing only from their random initialisations. They choose to combine the two vectors, as that gives a small performance boost, and this is also the default output of the training tool. Like the paper, I configured GloVe to output W+$\tilde{W}$.

5.3.2 Evaluation of the word embeddings

As was stated earlier, the intention of the enC³ word vectors is to be as comparable to the example GloVe word vectors as possible, as it is not feasible for this project to train more word vectors based on other corpora for comparison. This also means that the evaluation scores of the enC³ word vectors cannot be compared directly with word vectors created with other frameworks like *Word2Vec* or *fastText*, or even with GloVe vectors created with different parameters than these. The 'most optimal' word vectors for enC³ are in other words yet to be created, and will also depend on the task.

The evaluation scores of the GloVe example vectors are listed in the paper, but they are also available for download from the GloVe website. For the evaluation and comparison, I have run my own evaluations using two of the available sets of pre-trained word vectors, in addition to my own:

The first word vectors are trained on a Wikipedia 2014 dump (1.6B tokens) concatenated with the Gigaword Fifth Edition (4.3B tokens). Wikipedia data is also used by Bojanowski et al. (2016), and the Wikipedia + GigaWord combination is used by Fares et al. (2017). The latter find vectors trained on Wikipedia data to be especially good at the semantic analogy task. As the semantic analogy task tests for a lot of proper nouns like city and country names, the high

[4]https://groups.google.com/forum/#!msg/globalvectors/LHxRIWJnR6g/-IvY78hPNEAJ

performance of Wikipedia data on this task is expected, as it has particularly high coverage of those.

The second set of GloVe vectors are trained on Common Crawl data. From the GloVe paper and available documentation, it is unknown whether this data was taken from the WET files or extracted from HTML, the version(s) of the Common Crawl snapshot(s) used, and whether several months of data or only one monthly crawl was used. It seems unlikely that Pennington et al. (2014) have used a lot of time and resources on cleaning the data as extensively as described in this thesis, but it cannot be ruled out.

As of this writing, there is no information about the Common Crawl data set that was used at all, except for its size: 42 billion tokens. Information about how the data was obtained seems to be lost[5]. Since the best results of the evaluations of the GloVe model all came from the vectors trained on this data, this is unfortunate in terms of replicability.

The evaluation process

The same evaluation tasks that were described in section 5.2.3 will be used for this evaluation.

For the analogical similarity task, the same data set used by Mikolov et al. (2013) is used, similarly divided into semantic and syntactic parts. The evaluations are done using the Python evaluation code included in the GloVe-project, where vectors are normalised so that each vector has a length of 1 before the vector arithmetic is performed.

Evaluation of the different word similarity lists were done using the command line tools of Faruqui and Dyer (2014)[6]. The tools compute the Spearman's rank correlation coefficient between the human-rated lists taken as input and the ranked lists the tools produced by using the word vectors and cosine similarities.

However, for the word similarity task only, the above tools and Pennington et al. (2014) normalise their vectors in different ways, leading to different results. Both approaches use the cosine similarity, and thus inherently normalise so that each vector has a length of 1. However, Pennington et al. (2014, p. 8) state that "A similarity score is obtained from the word vectors by first normalising each feature across the vocabulary...", in addition to the normalisation done in cosine similarity calculation. I interpreted this as column-wise normalisation (if thinking of the set of word vectors as a matrix), where each feature is divided by the square root of the sum of its column's squared features – a sort of vector length normalisation, but using the column to calculate the norm instead of the row (the vector). Interestingly, the latter approach makes the similarity scores higher for

[5]https://github.com/stanfordnlp/GloVe/issues/74

[6]https://github.com/mfaruqui/eval-word-vectors

Model			Results		
Corpus	Dim.	Size	Semantic	Syntactic	Total
WikiGiga	300	6B	77.44	67.00	71.74
GloveCC	300	42B	**81.85**	69.25	74.96
enC³	300	135B	77.89	**73.14**	**75.30**

Table 5.4: The results of the analogy task, broken down to semantic and syntactic parts

all corpora and all tests, but does also introduce different relative results between the different corpora.

The performance boost with column-wise normalisation is noteworthy, and is as far as I know not widespread – I could not find any reports of it other than the short notice by Pennington et al. (2014). Both approaches were used for these evaluations, and as the former approach is the prevalent approach and the latter provides the best results, both results are reported below.

The analogy similarity results

The results of the analogy task can be seen in table 5.4. The most obvious result to note is that both models based on web corpora perform better than the model trained on the Wikipedia/Gigaword corpus. This can perhaps be an indication that size does matter. However, the data set the WikiGiga model is based on is highly dissimilar from the Common Crawl-based data sets of the two others, and attributing any performance difference to any specific reason is at this point not possible.

What is interesting is the break-down of the scores in their semantic and syntactic parts. The GloVe vectors based on Common Crawl data, or GloveCC, perform really well on the semantic task – a task Wikipedia based vectors normally perform well at. The enC³ model also outperforms the WikiGiga model in the semantic task, although not by much, and is beaten by around 4 percentage points by GloveCC. This is noteworthy, as the two corpora are based on the same type of data, and enC³ is over three times the size of GloveCC. With GloveCC's large performance increase over WikiGiga, one could have expected enC³ to either match or outperform GloveCC in this area.

Similarly interesting, enC³ outperforms the other two models in the syntactic part of the results. While GloveCC has a syntactic performance increase over the WikiGiga model, its largest increase was in the semantic results. It is not easy to attribute the performance gain of the enC³ model to some inherent characteristic of the corpus. The size difference can be the whole reason the results are better,

Model	WS353	MC	RG	SCWS	RW	SL999	YP	V144
WikiGiga	60.5	70.3	76.6	56.3	**41.2**[7]	37.1	56.1	30.5
GloveCC	62.9	**77.7**	**81.2**	54.1	38.5	**37.4**	53.2	32.8
enC³	**65.1**	76.7	78.6	**56.8**	39.4	37.1	**56.4**	**35.5**

Table 5.5: The results of the word similarity task with vector normalisation so that each vector has length 1

Model	WS353	MC	RG	SCWS	RW	SL999	YP	V144
WikiGiga	65.6	72.4	76.9	61.4	45.9	40.8	58.0	36.1
GloveCC	**75.7**	**83.6**	**82.3**	65.3	48.7	**45.3**	59.6	37.2
enC³	**75.7**	79.7	78.0	**66.1**	**50.1**	42.3	**59.8**	**37.8**

Table 5.6: The results of the word similarity task where each feature is normalised across the vocabulary.

or the performance increase may come from the corpus cleaning.

In total, the enC³ model performs slightly better than the GloveCC model, and a lot better than the WikiGiga model.

The word similarity results

The word pair lists that were chosen are listed and described below. They have either been created using different methods, or are trying to test for different characteristics. Some tests are included because they are popular, and typically used for the word similarity task. All of the lists have had their word pairs scored by a number of human judges, and the score used to rank the word pairs is the average of the human scores. The scales used are different for the different word pair lists, but all of the lists have asked their raters to rate word pairs on a scale from non-similar to similar, with some lists like SimLex-999 giving additional instructions.

WS353 was prepared by Finkelstein et al. (2001), and contains 353 noun pairs that have had their similarities judged by humans, 153 of them by 13 judges and the other 200 by 16 judges.

MC contains 30 human annotated noun pairs that were originally prepared to show the relationship between semantic similarities and similar contexts (Miller

[7]The out of vocabulary rate is very high, at 12.4%, so the score of 41.2 is not comparable to the others. The oov rate was 0.9% for enC³ and 1.0% for GloveCC.

and Charles, 1991).

RG is similar in intention to the MC data set, and contains 65 noun word pairs (Rubenstein and Goodenough, 1965). Both MC and RG are small and older datasets, and it is questionable whether they are large enough to give an accurate account of performance in the word similarity task. Small rank differences using lists of such a small size can give large score differences, and it can be argued that high correlations with the larger lists should count for more. However, as these lists are prevalent when the word similarity task is performed, I'm including them for relatability and comparability.

SCWS – or Stanford's Contextual Word Similarities – was an answer to the word pair lists above. They state that it is unclear how assigning similarities to words in isolation, without the context, affects the scores human judges set on word pairs. In SCWS, they provide the context and POS-tag together with the words, and let 10 human raters score the similarity of the word pair. It is important to note that there are 10 human raters per word pair, and not 10 individuals judging all the word pairs. They did this by using Amazon Mechanical Turk, and letting highly rated non-experts rate the similarity, citing the experiments of Snow, O'Connor, Jurafsky, and Ng (2008), who showed that non-expert annotators can achieve very close inter-annotator agreement with expert raters. The dataset has 1328 noun–noun-pairs, 399 verb–verb, 140 verb–noun, 97 adjective–adjective, 30 noun–adjective and 9 verb–adjective pairs, for a total of 2003 word pairs (Huang, Socher, Manning, and Ng, 2012).

RW (Rare Word) specifically tests for the embeddings' ability to model rare words, by preparing 2034 word pairs collected from different frequency ranges, thus making sure rare words are present (Luong, Socher, and Manning, 2013). These word pairs are rated in the same way as the SCWS data set, using Amazon Mechanical Turk and collecting 10 ratings per word pair. Because of the nature of rare words, many of these words are out of the vocabulary of popular word vectors, and this is one of the tests where corpus size probably affects the out-of-vocabulary rate directly.

SimLex-999 is different from several of the other word pair lists, because it explicitly differentiates words that are *similar* (coast–shore) with words that are *associated*, but not similar (clothes–closet), giving high scores to the former, and low to the latter (Hill, Reichart, and Korhonen, 2016). The popular WS353 list described above gives high scores to both of these examples. SimLex-999 is rated by 500 human raters, also using Amazon Mechanical Turk, where each word pair was on average rated by 50 raters. The data set contains 666 noun–noun pairs, 222 verb–verb and 111 adjective–adjective, for a total of 999 word pairs.

YP was made by Yang and Powers (2006), who made their own verb–verb word pair list, containing 130 verb pairs, because most datasets at the time mostly had noun pairs. The word pairs were rated by 2 academic staff and 4 postgraduate students.

V144 – or the Verb-144 dataset – is in the same spirit as the YP dataset. Like the name implies, the dataset contains 144 verb pairs scored by 10 native English speaking people (Baker, Reichart, and Korhonen, 2014).

The two sets of results show non-trivial differences between which model that performs the best, and as with other areas of NLP, performance is task-specific: Good relative performance of a model on one word pair list does not necessarily equate to good performance on another list.

The results are presented in both table 5.5 and in table 5.6. As mentioned earlier, there are two sets of results of the word similarity task, because of the different way Pennington et al. (2014) normalise the vectors. In table 5.5, the results of the task with 'regular' length normalisation are presented, while table 5.6 has the results of the feature normalisation – where each feature is normalised across the vocabulary, column for column, in addition to the length normalisation. Interestingly, this additional layer of normalisation makes a larger difference in the results of this task than the choice of corpus for most of the datasets.

There are large differences between the two ways of normalising, where every single score improves from doing length normalisation to feature normalisation. The differences between the models are also affected by the different normalisation, and in some cases this change in difference affects which model performs better than the others: From enC³ scoring the highest with the WS353 list when length normalised, GloveCC matches the score when feature normalising. The Rare Word list favor the WikiGiga model (although the out of vocabulary-rate is a lot higher than the two other models) with length normalisation, but favor enC³ with feature normalisation. Some differences become larger, like the MC, RG and SimLex-999 scores being even more in favor of GloveCC, and some become smaller, like YP and Verb-144.

Even when performing the feature normalisation and obtaining the higher results, some of the results do not match the results of Pennington et al. (2014), while others do. In particular, the scores using the RG, SCWS and RW lists deviate from the results listed by the paper, while the scores from using WS353 and MC are almost the same (a variation of 0.2 and 0 percentage points, respectively). This is an issue of replicability, and while regrettable, does not affect these comparisons, as all the corpora were tested using the same code and datasets. The code, created by Faruqui and Dyer (2014) and modified by me to achieve the extra feature normalisation, is available online[8].

The scores do not give a clear picture for which model that performs best, but do clearly show that the WikiGiga model is outperformed in all cases. Like in the analogy task, it is difficult to point to why this is, but it seems unlikely that web data would be inherently better suited for this task, and an argument can perhaps be made that the size difference matters. The larger word pair lists all favor enC³, except for SimLex-999, that have more advanced notions of similarities. By simply counting how many word pair lists favor the different models, however, enC³ achieves the best score for most lists.

5.3.3 Better intrinsic results: A better corpus?

The enC³ model can be said to have performed marginally better than the Common Crawl model used in the GloVe paper and clearly better than the Wikipedia/GigaWord-model that is typically used in intrinsic evaluations of word embeddings. This is exciting, and might be an indicator that larger corpora (and perhaps cleaned corpora) are good sources of data for these tasks.

The comparison done in this chapter was done by training the enC³ word embeddings using as similar parameters to the GloVe embeddings as possible. This was done to make sure that the prospective performance differences came as a result of the differences in corpus quality, and not from choosing parameters more suited to the chosen tasks. However, the disadvantage of this approach is that the potential of the enC³ corpus might have been impeded by not choosing the parameters most suited for the corpus. For instance, Pennington et al. (2014) increase vocabulary size as corpus size increases. Other parameters might also affect the performance of the word vectors because of corpus size. As an example, it seems likely that higher vector dimensions are more suited to exploit large corpus sizes. As a result, the best comparison of corpora using word embeddings would probably be to tweak the parameters individually for the different corpora so that each corpus performs the best for the chosen task, and then compare the results. This is a time-consuming endeavour, however, seeing as the training of one set of word vectors from enC³ took over a week.

Another question is whether or not the results of these intrinsic word vector evaluations actually give an indication of corpus quality outside of the tasks that were performed. For instance, Chiu, Korhonen, and Pyysalo (2016) show in their paper that there is no correlation between the performance of the word similarity task and the performance of their three chosen downstream tasks (Named Entity Recognition, PoS-tagging and chunking), except for one of the word pair lists: SimLex-999.

This might only mean that word vectors specifically tuned to perform well

[8]https://github.com/kjetilbk/enc3tools

in different word similarity tasks do not necessarily perform well in other downstream tasks, and does not necessarily say much about the corpus used to train the embeddings. It would be interesting to do similar extensive research using different corpora with different word vector training parameters on several downstream tasks, to see which corpora perform the best, and whether or not their performance is task-specific.

Chapter 6

Conclusion and future work

In this project, I have constructed a large collection of texts from web data provided by the Common Crawl Foundation. The 135 billion word corpus was constructed using *texrex*, an open-source toolchain that combines several cleaning tasks to produce a high-quality corpus from noisy web documents. While several web corpora exist, there are none at the scale of this thesis' **En**glish **C**ommon **C**rawl **C**orpus – or *enC³*. The results of the corpus evaluation showed large improvements over the much smaller, more typically used Wikipedia+Gigaword corpus, and incremental improvements over a web corpus at a third the size used as comparison. This indicates that size does matter, although it is not the sole important factor of corpus quality.

The corpus was constructed using data from the August 2016 Common Crawl, and chapter 3 provided the first systematic review of said data, where the quality of the provided extracted text files was also assessed and deemed insufficient. Specifically, the text extraction itself – the removal of HTML tags and entities, as well as fixing broken encoding – was incomplete, and would have needed to be redone. The low quality of the text extraction left the files containing archived web data as the best source of data for the corpus construction. The extracted text files were compared to the text extraction performed by texrex, which showed that texrex cleaned the data much better than what has been done with the extracted text files. All the steps required for high-quality text extraction thus had to be performed from scratch. Additionally, language identification was performed on the Common Crawl data, identifying the low amount of data from languages other than English, making it difficult to construct large corpora for them.

Discussion about the desiderata of a corpus construction pipeline, before concluding with and presenting the texrex pipeline was done in chapter 4. Then, the particular configuration of the tools that I chose for the current work were discussed. The corpus construction was configured to achieve high quality, but also to run efficiently, possibly removing non-negligible amounts of data to achieve acceptable run times. A large challenge of the process was handling the extensive amounts of data, which elicited these kinds of choices in the configuration. Next, the chapter gave a detailed account of how the jobs were set up to run on a cluster, focusing on replicability, due to the desire to possibly repeat the process in the future (see future work below). The result was the first representation of the enC³ corpus: the XML representation. The post-processing of these resulting corpus files was done in two steps. As texrex performs non-destructive boilerplate detection, the first step was to provide a boilerplate-free text representation of the corpus by developing a tool called xml2txt. The second step provided the third and final corpus representation, where Stanford CoreNLP was used to produce tokenised and sentence-segmented CoNLL files. Given the corpus size, such post-processing steps are not trivial to perform. By making enC³ available in all of these representations, the corpus is easy to use immediately for downstream tasks, while still preserving as much data as possible, should a user wish to perform post-processing using different parameters than I used.

The evaluation of the corpus was performed by training word vectors using GloVe with the corpus as input, and was described in chapter 5. The parameters used for this training were as close to the ones used by Pennington et al. (2014) as possible, seeing as the intention was not to train the best possible word embeddings for the chosen tasks, but to give as much of an apples-to-apples comparison as possible. The word vectors were compared with two of the pre-trained word vectors available from GloVe, the first trained on a 6 billion token Wikipedia+GigaWord corpus and the second trained on a 42 billion token web corpus also based on Common Crawl data.

Two tasks were performed to evaluate the word vectors: A word analogy task using the standard dataset provided by Mikolov et al. (2013) and a word similarity task, where the latter was performed with respect to a large suite of standard benchmark datasets for word similarity. The findings were interesting in several regards. For one, a significant performance boost was achieved by column-wise normalisation of the vectors. Furthermore, both the web corpora considerably outperformed the smaller Wikipedia/News corpus, and while the enC³ model did largely outperform the smaller web corpus, the manner in which it did so was unexpected. The smaller web corpus outperformed the enC³ model in the semantic part of the analogy task by a substantial margin, as well as in the word similarity task using the more complex SimLex-999 word pair list. It is clear that while corpus size is an important factor for word vector performance, it is not the only one at play.

This work has investigated how to construct a corpus from web data at a large scale, and has evaluated the resulting corpus using the popular task of training word vectors, which result is used for a large number of tasks in NLP. As such, it should be noted that the word vectors themselves provide a valuable asset to the NLP community, and I will make the pre-trained enC³ embeddings available through the community repository launched by Fares et al. (2017), open for non-commercial researchers in the Nordic region. The results of the thesis are concrete contributions to the NLP community:

- enC³ – a very large web corpus, available through the mentioned community repository.
- The word vectors trained on enC³ – also available through said repository.
- A detailed account of how to perform web corpus construction using texrex, if one wants to replicate the process.
- A systematic review of the WET files provided by the Common Crawl, and their shortcomings.
- The tools developed for analysing Common Crawl files, and for extracting non-boilerplate text from the corpus files, made publicly available[1].

6.1 Future work

Several parts of this thesis can be fleshed out and explored further in future work. This relates to issues that either manifested themselves as a consequence of questions that were revealed after observing the results, or would have required too much time to perform within the constraints of this thesis. Some such directions for future work are summarised below.

6.1.1 Replicating the process

Using the work of this thesis and the tools provided with it, a goal for the Language Technology Group at the University of Oslo is to replicate the process and construct an updated web corpus from Common Crawl data once a year. Seeing how Common Crawl data has developed and increased in size from when this thesis was begun – with the compressed WARC files in August 2016 taking up 30TB, and 60TB in the March 2017 crawl – there is absolutely potential for large, clean web corpora to be produced each year.

6.1.2 Researching the impact of boilerplate

Much of the text that was removed in the cleaning process was boilerplate, because of the assumption of boilerplate impeding the quality of the corpus. There are several reasons why it would be interesting to test that assumption, however. The results of the word vector evaluation showed that in a few areas, the smaller web corpus performed better than enC³. There is no knowing how that corpus was constructed, but I assume thorough cleaning like it was done with enC³ was

[1]https://github.com/kjetilbk/enc3tools

not performed. It is hard to know exactly what the practical implications are for various use cases, but being able to account for the impact of boilerplate on task results could help explain some of the results.

Additionally, even though the boilerplate removal performs well, it will never be perfect and will sometimes remove paragraphs that are non-boilerplate content. If it appears that boilerplate has little impact on corpus quality, these removed paragraphs of text could have increased corpus size, and as an extension, impacted results of tasks using the corpus.

The XML representation of the enC³ corpus – where boilerplate detection was performed in a non-destructive manner, making boilerplate still possible to extract – suits this task well. Extracting two text sub-corpora from this representation – one with a lot of boilerplate and one with very little – is trivial, and would provide a good basis for measuring the impact of boilerplate.

6.1.3 Investigating overlap between monthly crawls

The Common Crawl Foundation provides monthly crawls, but it is not clear to what degree the different crawls overlap. There probably is a lot of duplication between the August and the September crawls, but what about crawls several months, or even a year apart? This investigation can lead to knowing whether or not it is worth downloading several crawls and create a multi-month corpus, possibly increasing the size of enC³, or making C³ of other languages possible.

6.1.4 noC³?

Including the possibilities that open up if the overlap question is clarified, the Common Crawl Foundation announced with the release of their December 2016 crawl[2] that future crawls would have greater coverage of multi-lingual content. Even if processing multiple monthly crawls is deemed too costly, the increased non-English support could prove to be enough to construct corpora for under-resourced languages like Norwegian.

6.1.5 Evaluation with more tasks

As corpora are used for so many tasks – including training word vectors that are also used for a large number of tasks – it would be preferable with a fuller understanding of what a good corpus for different types of tasks is. Evaluation of constructed corpora using more downstream tasks can possibly give some of that

[2]http://commoncrawl.org/2016/12/december-2016-crawl-archive-now-available/

understanding, as the results for each task are interesting in and of themselves, and perhaps they together will provide an indication of some common factors that contribute to good performance in several downstream tasks.

Text types, domain and genre also play a part in how well a corpus performs for different tasks, and for processing of general web text a large web corpus might be exactly what is desired.

Bibliography

Baker, S., Reichart, R., & Korhonen, A. (2014). An Unsupervised Model for Instance Level Subcategorization Acquisition. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (pp. 278–289). Doha, Qatar.

Banko, M. & Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics* (pp. 26–33). Toulouse, France.

Baroni, M. & Bernardini, S. (2004). BootCaT: Bootstrapping Corpora and Terms from the Web. In *Proceedings of the ninth international conference on Language Resources and Evaluation*. Lisbon, Portugal.

Baroni, M., Bernardini, S., Ferraresi, A., & Zanchetta, E. (2009). The WaCky Wide Web: A collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, *43*(3), 209–226.

Baroni, M., Chantree, F., Kilgarriff, A., & Sharoff, S. (2008). Cleaneval: A competition for cleaning web pages. In *Proceedings of the sixth international conference on Language Resources and Evaluation*. Marrakech, Morroco.

Baroni, M., Dinu, G., & Kruszewski, G. (2014). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 238–247). Baltimore, Maryland, USA.

Bauer, D., Degen, J., Deng, X., Herger, P., Gasthaus, J., Giesbrecht, E., ... Evert, S. (2007). FIASCO: Filtering the Internet by Automatic Subtree Classification, Osnabruck. In *Proceedings of the 3rd Web as Corpus Workshop* (Vol. 4, pp. 111–121). Louvain-la-Neuve, Belgium.

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, *3*, 1137–1155.

Bernardini, S., Baroni, M., & Evert, S. (2006). A WaCky introduction.

Biemann, C., Bildhauer, F., Evert, S., Goldhahn, D., Quasthoff, U., Schäfer, R., ... Zesch, T. (2013). Scalable Construction of High-Quality Web Corpora. *Journal for Language Technology and Computational Linguistics*, *28*(2), 23–59.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*.

Broder, A. Z. (1997). On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences* (pp. 21–29). Washington DC, USA.

Chiu, B., Korhonen, A., & Pyysalo, S. (2016). Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance. (pp. 1–6). Berlin, Germany.

Collobert, R. & Weston, J. (2008). A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th international conference on Machine learning* (pp. 160–167). Helsinki, Finland.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (almost) from Scratch. *Journal of Machine Learning Research*, *12*, 2493–2537.

Fares, M., Kutuzov, A., Oepen, S., & Velldal, E. (2017). Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*. Gothenburg, Sweden.

Faruqui, M. & Dyer, C. (2014). Community evaluation and exchange of word vectors at wordvectors.org. In *Proceedings of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland, USA.

Ferraresi, A., Zanchetta, E., Baroni, M., & Bernardini, S. (2008). Introducing and evaluating ukWaC, a very large web-derived corpus of English. In *Proceedings of the 4th Web as Corpus Workshop* (pp. 47–54). Marrakech, Morocco.

Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., & Ruppin, E. (2001). Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web* (pp. 406–414). Hong Kong, China.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*.

Goldberg, Y. (2015). A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research*, *57*, 345–420.

Habernal, I., Zayed, O., & Gurevych, I. (2016). C4Corpus: Multilingual Web-size Corpus with Free License. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation* (pp. 914–922). Portorož, Slovenia.

Harris, Z. S. (1954). Distributional structure. *Word*, *10*(2-3), 146–162.

Hill, F., Reichart, R., & Korhonen, A. (2016). SimLex-999: Evaluating Semantic Models with (Genuine) Similarity Estimation. *Computational Linguistics*.

Huang, E. H., Socher, R., Manning, C. D., & Ng, A. Y. (2012). Improving Word Representations via Global Context and Multiple Word Prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers* (Vol. 1, pp. 873–882). Jeju Island, South Korea.

International Organization for Standardization. (2009). ISO 28500-2009, Information and documentation – WARC file format.

Jakubiček, M., Kilgarriff, A., Kovař, V., Rychly, P., & Suchomel, V. (2013). The TenTen corpus family. In *Proceedings of the 7th International Corpus Linguistics Conference* (pp. 125–127). Valladolid, Spain.

Jones, R. & Ghani, R. (2000). Automatically building a corpus for a minority language from the web. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics* (Vol. 38, pp. 29–36). Hong Kong, China.

Keller, F. & Lapata, M. (2003). Using the Web to Obtain Frequencies for Unseen Bigrams. *Computational Linguistics*, *29*(3), 459–484.

Kilgarriff, A. (2007). Googleology is bad science. *Computational Linguistics*, *33*(1), 147–151.

Kilgarriff, A. & Grefenstette, G. (2003). Introduction to the special issue on the web as corpus. *Computational Linguistics*, *29*(3), 333–347.

Kilgarriff, A., Rychlý, P., Jakubicek, M., Kovář, V., Baisa, V., & Kocincová, L. (2014). Extrinsic Corpus Evaluation with a Collocation Dictionary Task. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*. Reykjavik, Iceland.

Kohlschütter, C., Fankhauser, P., & Nejdl, W. (2010). Boilerplate detection using shallow text features. In *Proceedings of the third international conference on Web search and data mining* (pp. 441–450). New York City, USA.

Lui, M. & Baldwin, T. (2012). Langid.Py: An Off-the-shelf Language Identification Tool. In *Proceedings of the Association for Computational Linguistics: System Demonstrations* (pp. 25–30). Jeju Island, South Korea.

Luong, M.-T., Socher, R., & Manning, C. D. (2013). Better Word Representations with Recursive Neural Networks for Morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning* (pp. 104–113). Sofia, Bulgaria.

Luotolahti, J., Kanerva, J., Laippala, V., Pyysalo, S., & Ginter, F. (2015). Towards Universal Web Parsebanks. *Proceedings of the Third International Conference on Dependency Linguistics*, 211–220.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of the Association for Computational Linguistics: System Demonstrations* (pp. 55–60). Baltimore, Maryland, USA.

McEnery, T. & Wilson, A. (2001). *Corpus Linguistics: An Introduction*. Edinburgh University Press.

Mihalcea, R. & Moldovan, D. I. (1999). A method for word sense disambiguation of unrestricted text. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics* (pp. 152–158). College Park, Maryland, USA.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Miller, G. A. & Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and cognitive processes*, *6*(1), 1–28.

Parker, R., Graff, D., Kong, J., Chen, K., & Maeda, K. (2011). *English Gigaword Fifth Edition*. Technical Report. Linguistic Data Consortium, Philadelphia.

Pasternack, J. & Roth, D. (2009). Extracting article text from the web with maximum subsequence segmentation. In *Proceedings of the 18th international conference on World wide web* (pp. 971–980). New York, USA.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (Vol. 14, pp. 1532–1543). Doha, Qatar.

Pomikálek, J. (2011). *Removing boilerplate and duplicate content from web corpora* (Doctoral thesis, Masaryk University, Faculty of Informatics, Brno).

Pomikálek, J., Jakubícek, M., & Rychlý, P. (2012). Building a 70 billion word corpus of English from ClueWeb. In *Proceedings of the eighth international conference on Language Resources and Evaluation* (pp. 502–506). Istanbul, Turkey.

Resnik, P. (1999). Mining the web for bilingual text. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics* (pp. 527–534). College Park, Maryland, USA.

Rubenstein, H. & Goodenough, J. B. (1965). Contextual Correlates of Synonymy. *Communications of the ACM*, *8*(10), 627–633.

Schäfer, R. (2014). *texrex*. The manual for the texrex toolchain.

Schäfer, R. (2015). Processing and querying large web corpora with the COW14 architecture. In *Proceedings of the 3rd Workshop on Challenges in the Management of Large Corpora* (pp. 28–34). Lancaster, United Kingdom.

Schäfer, R. (2016a). Accurate and efficient general-purpose boilerplate detection for crawled web corpora. *Language Resources and Evaluation*. online first.

Schäfer, R. (2016b). CommonCOW: Massively Huge Web Corpora from CommonCrawl Data and a Method to Distribute them Freely under Restrictive EU Copyright Laws. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation*. Portorož, Slovenia.

Schäfer, R. (2016c). On Bias-free Crawling and Representative Web Corpora. In *Proceedings of the 10th Web as Corpus Workshop* (pp. 99–105). Berlin, Germany.

Schäfer, R., Barbaresi, A., & Bildhauer, F. (2013). The Good, the Bad, and the Hazy: Design Decisions in Web Corpus Construction. In *Proceedings of the 8th Web as Corpus Workshop* (pp. 7–15). Lancaster, United Kingdom.

Schäfer, R., Barbaresi, A., & Bildhauer, F. (2014). Focused Web Corpus Crawling. In *Proceedings of the 9th Web as Corpus workshop* (pp. 9–15). Gothenburg, Sweden.

Schäfer, R. & Bildhauer, F. (2012). Building Large Corpora from the Web Using a New Efficient Tool Chain. In *Proceedings of the Eight International Con-*

ference on Language Resources and Evaluation (pp. 486–493). Istanbul, Turkey.

Snow, R., O'Connor, B., Jurafsky, D., & Ng, A. Y. (2008). Cheap and fast—but is it good?: Evaluating non-expert annotations for natural language tasks. In *Proceedings of the conference on Empirical Methods in Natural Language Processing* (pp. 254–263). Honolulu, Hawaii.

Spearman, C. (1904). The proof and measurement of association between two things. *The American Journal of Psychology*, *15*(1), 72–101.

Spousta, M., Marek, M., & Pecina, P. (2008). Victor: The web-page cleaning tool. In *Proceedings of the 4th Web as Corpus Workshop* (pp. 12–17). Marrakech, Morroco.

Suchomel, V. & Pomikálek, J. (2012). Efficient web crawling for large text corpora. In *Proceedings of the seventh Web as Corpus Workshop* (pp. 39–43). Lyon, France.

Ueyama, M. (2006). Evaluation of Japanese web-based reference corpora: Effects of seed selection and time interval. *WaCky! Working papers on the Web as Corpus*, 99–126.

Versley, Y. & Panchenko, Y. (2012). Not Just Bigger: Towards Better-Quality Web Corpora. In *Proceedings of the 7th Web as Corpus workshop*. Lyon, France.

Yang, D. & Powers, D. M. W. (2006). Verb similarity on the taxonomy of wordnet. In *In the 3rd International WordNet Conference*. Jeju Island, Korea.