UNIVERSITY OF OSLO
Department of Informatics

A Corpus Builder for Wikipedia

Master's thesis

Lars Jørgen Solberg

November 15, 2012

Abstract

We present in this work a method of creating high-quality corpora from collections of user generated content, which we apply on a snapshot of Wikipedia to create a very large corpus. Both our software implementation and the corpus are released to the public. Our approach makes use of both machine learning and hand-written rules to remove a large portion of content that have little value for most information retrieval on natural language processing tasks. This work also contains a survey of several state of the art sentence boundary detectors and we develop methods of improving their performance by taking advantage of layout information. Finally, we perform a quantitative comparison with a corpora created with an earlier tool.

Acknowledgements

My gratitude goes out to my advisers Stephan Oepen and Jonathon Read for their encouragement and thoughtful guidance throughout this project.

I would also like to thank my fellow students for their enjoyable company and valuable input: Emanuele Lapponi, Lars-Erik Bruce, Murhaf Fares, Sindre Wetjen, Johan Benum Evensberget, Charlotte Løvdahl and Arne Skjærholt.

Contents

Contents 5

List of Figures 7

List of Tables 8

List of Listings 9

1 Introduction 11
- 1.1 Problem Definition 12
- 1.2 Background: Clean and Dirty Text 15
- 1.3 Thesis Overview 18
- 1.4 Summary of Main Results 19

2 Background and Motivation 21
- 2.1 Format and Structure of Wikipedia 21
- 2.2 Previous Work 31
- 2.3 Tools for processing Wikipedia Dumps 33

3 Article Extraction and Parsing 37
- 3.1 Choosing a Wiki Parser 38
- 3.2 Markup Extraction 39
- 3.3 Templates 40
- 3.4 Section Identification 50

4 Content Selection 53
- 4.1 Hand-Crafted Rules vs. Machine Learning 54
- 4.2 Background: N-gram Models 55
- 4.3 Previous Work 57
- 4.4 Our Approach 60
- 4.5 Finding the Optimal Configuration 65
- 4.6 Revisiting Relevant Linguistic Content 81

5 Sentence Segmentation **85**
5.1 Choosing a Sentence Segmenter 86
5.2 Fine Tuning 92
5.3 Restoring Markup 94

6 The Corpus **99**
6.1 GML 100
6.2 Corpus Generation and Structure 103
6.3 Evaluation and Comparison with WikiWoods 1.0 104

7 Conclusion **111**
7.1 Future Work 113

Glossary **117**

Bibliography **121**

A Elements of GML and Wiki Markup **127**

B Template Lists **131**
B.1 Most Used Templates 131
B.2 Template Naming Conventions 144

List of Figures

1.1 Parts of the article “Context-free grammar” 14
1.2 Overview of our system 18

2.1 Wikipedia’s page editor 23
2.2 A simple page in Wikipedia 24
2.3 Pictures from the article “Albert Einstein” 26

3.1 Overview of our system 37
3.2 Distribution of templates 43
3.3 Article structure before and after section identification 50

4.1 Overview of our system 53
4.2 Hand crafted rules vs. machine learning 55
4.3 Perplexity on test_A 75
4.4 F_1-score on test_B 75
4.5 F_1-score on the gold standard 76
4.6 F_1-score on the silver standard 77

5.1 Overview of our system 85

6.1 Overview of our system 99
6.2 Article and Sentence identifiers 104
6.3 Sentence distribution by length 108
6.4 Parsing coverage by sentence length 109

7.1 Overview of our system 112

List of Tables

2.1 Namespaces 22

3.1 Comparison of dumpHTML and mwlib 38
3.2 Most included templates 44
3.3 Template naming conventions 46

4.1 Top-five performers in the “Text-only” part of CleanEval 57
4.2 Training and test sets 68
4.3 Significance test on the gold standard 79
4.4 Significance tests on the silver standard 79
4.5 Classifier performance with the original and refined heuristics . 80

5.1 Candidate systems 87
5.2 Results from our earlier experiments 87
5.3 Performance of sentence boundary detectors 91
5.4 Effect of different ad-hoc rules 92

6.1 Figures from WikiWoods 1.0 and WikiWoods 2.0 105
6.2 Figures from the samples 107

A.1 Elements of wiki markup 127

B.1 Most included templates 131
B.2 Template naming conventions 144

List of Listings

2.1 The definition for “Flag” 28
2.2 Snippet from the source code for the template “Fb r” 29
2.3 The expansion of `{{Flag|China}}` 30
2.4 One sentence from WeScience 32
2.5 Excerpt from WikiWoods 35

3.1 Marks from `{{Flag|China}}` 48
3.2 Marks from `{{Flag|China}}` after modifying mwlib 49

4.1 A section after preproccesing. 62
4.2 Telnet session with a SRILM server. 64
4.3 The sign test implemented in Python 72
4.4 A clean section 81
4.5 A section containing both clean and dirty text 82

5.1 `build_tokens()` 95
5.2 `markup_sentences()` 96

6.1 GML sample 100
6.2 The article “`"J"` Is for Judgment” 105
6.3 The article “Giant Steps (disambiguation)” 106

Chapter 1

Introduction

Wikipedia is a free, on-line encyclopedia that currently has over 4 million articles in its English version.[1] It covers a broad range of topics and the quality of the writing is on the whole better than most other user generated content. There are several localised versions, and articles are often explicitly linked to their counterpart in other languages. Its great size is the result of user contributions, where visitors are encouraged to create and improve articles.

Articles are internally written in a markup language, dubbed wiki markup. The language is designed to be easy to learn and unobtrusive, many of the directives resemble conventions occasionally used when text formatting is unavailable. Modifications to Wikipedia articles take effect immediately.

Markup languages are used to assign properties to different parts of documents. This is usually done in order to specify the appearance (e.g. This text should be typeset in *italics*) of different parts of a document, but the assigned property can in principle be anything. Some directives of wiki markup have a mostly visual effect, while others are used to assign properties that are meaningful when interpreting the text, like: "this text is in Spanish" or "this is an abbreviation".

Its size combined with its permissive license,[2] gives Wikipedia the potential to be an important resource for the natural language processing (NLP) and information retrieval (IR) communities. Unfortunately due to its internal format, Wikipedia does not easily lend itself as source material for such tasks. One obstacle is the structure of the markup language, which

[1] `http://www.wikipedia.org`

[2] CC-BY-SA 3.0 Unported License available at `http://en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License`

despite its low barrier to entry for humans, has some features that make it best viewed as a programming language. Due to how the different markup directives interact with each other, a naive strategy of simply ignoring the more complex elements will result in invalid markup.

There is also the problem of identifying which content is relevant for further analysis, as there is no clear segregation of the main content from meta-information, navigation elements and so on. The elements that typical NLP or IR applications will not be interested in are usually referred to as *noise* and often appear in the form of navigational aids (e.g. "For other uses, see Tail (disambiguation).") or meta information (for instance the often seen "[citation needed]"), but there are many other types of noise in the above informal sense.

This chapter provides an outline for this thesis and an overview of a system we have developed for extracting the linguistic content from Wikipedia that is relevant for common NLP and IR tasks.

1.1 Problem Definition

In this thesis we will develop a method of building corpora by extracting the textual content from the English Wikipedia. We will implement this method and release the software and the corpus. The content we extract is enriched with selected markup elements that are potentially useful when creating NLP and IR systems. We also aim to filter out as much of the non-interesting content as possible. As we will describe in the following chapters, making the distinction between interesting and non-interesting content is not always easy, neither on the conceptual level of defining what constitutes "interesting" and "non-interesting" nor on the implementation level.

Our system is in many ways intended to be an improved version of Corpus Clean (Ytrestøl, 2009), an earlier effort on creating a corpus builder for Wikipedia that was used in the creation of the WeScience and WikiWoods (Ytrestøl et al., 2009; Flickinger et al., 2010) corpora. Like our system, Corpus Clean preserves the markup directives that were deemed relevant for further linguistic analysis while discarding others. However, Corpus Clean is unable to interpret some of the more complex directives of wiki markup, for instance template inclusions. The results of improper template handling can be observed by looking for fragments of wiki markup in WikiWoods. Corpus Clean had an heuristic approach to page cleaning and discarded article sections based on their heading, as a result of this both WikiWoods and WeScience contain more noise than necessary. We

will come back to Corpus Clean (Section 2.3.3) and the corpora created with it (Section 2.2.2).

Wikipedia is a rich resource of structured data, however the use of "textual content" above means that we will have a NLP centred approach. We will not make any effort to retrieve structured information for instance by exploiting the inter article link structure or parsing "information boxes" (tabular structures with short labeled phrases) and the like. There are several interesting initiatives in this field as for instance "DBpedia" (Auer et al., 2007).

1.1.1 Relevant Linguistic Content

Unfortunately, most Wikipedia articles have some content that would contribute very little, if not be out right detrimental, to any downstream usage of our corpus.

CleanEval (a shared task discussed in Section 1.2) used an allegory of cleaning for the process of removing unwanted content from web pages. Using the term "dirt" when referring to unwanted content follows naturally from use of the word "cleaning". Alternatively, if one use the metaphor of detecting a signal (the relevant linguistic content, one can use the term "noise" for the unwanted content. There are several varieties of noise: such as navigational aids, meta information, non-textual content or textual content with no grammatical structure.

A notion related to that of noise is "boilerplate". Wikipedia itself describes boilerplate as: "any text that is or can be reused in new contexts or applications without being changed much from the original.".[3] In the context of web content it takes on a slightly broader meaning of frequently repeated, mostly auto-generated, content like copyright notices, navigation bars and so on. Some, but not all, of the content that we consider dirty is boilerplate in this sense. For instance collections links, bibliographies and such are usually not considered boilerplate, but fall within our definition of noise. As we will see below the term is often used to refer to all unwanted content.

The content we *do* wish to retrieve are spans of text that contain information about the subject matter of the article and that have a form that requires grammatical analysis for interpretation. Such content will be described as "clean" or "relevant linguistic content".

Figure 1.1 shows three parts of the article "Context-free grammar" as it appeared on 15 September 2012. The parts that are highlighted do not

[3] http://en.wikipedia.org/wiki/Boilerplate_(text)

Figure 1.1: Parts of the article "Context-free grammar"

meet our criteria for relevant linguistic content. Those in gray are not the result of markup directives and no special processing are needed to remove them.

1 The header, left-hand navigation bar and the footer (not shown) are generated by the Mediawiki server and are not included in the wiki markup.

2 All articles have their title as their top level heading — this is not explicitly present in the article source. We generate this and insert it into the corpus.

3 This is a "message box" that contains meta information, and we consider these to be noise. It it is created by the inclusion of the template "Refimprove" in the article source. Templates are pages that are

intended to be included in other pages. Some of them simply contain static markup that is shown at the place of their inclusion, while others make use of the more complex features of wiki markup. Our approach to templates is described in Section 2.1.3.

4 Like the navigational frame, the table of contents is not in the article source, but auto-generated from the article structure and not something we want in the corpus.

5 These are "language links" that link to articles of the same topic in other languages, they are rendered on the navigation bar instead of where they appear in the markup. They are navigational elements and are removed.

6 Footnotes are usually used for bibliographic references, but are also used for regular footnotes. All footnotes are removed.

7 These links lead to the page editor and are not present in the article source.

8 Most articles have one or more sections for references, footnotes, external links etc. that contain little relevant linguistic content. The identification of such sections cannot be done by simply looking at the markup, our approach to is discussed later in Chapter 4.

9 A "navigation box", this is a navigational aid that links to articles with related topics. It is not uncommon for an article to have one or more navigation boxes, these are usually found near the end of an article. As with all other navigational elements, we consider them noise.

1.2 Background: Clean and Dirty Text

There has been several earlier efforts on extracting content from noisy web documents. Below we will look at how some earlier efforts defined the classes clean and dirty text.

1.2.1 CleanEval

The CleanEval shared task was held in 2007 with the objectives of (a) removing boilerplate from arbitrary web pages and (b) recovering some of the basic page structure (Baroni et al., 2008). In preparation for this task 741 English and 713 Chinese web pages were manually cleaned. The

annotator guidelines that were used are summarised as follows in Pomikálek (2011, p. 21):

> In short, the CleanEval guidelines instruct to remove boilerplate types such as:
>
> - Navigation
> - Lists of links
> - Copyright notices
> - Template materials, such as headers and footers
> - Advertisements
> - Web-spam, such as automated postings by spammers
> - Forms
> - Duplicate material, such as quotes of the previous posts in a discussion forum

Some of these boilerplate types are not a problem when cleaning Wikipedia articles: The navigation bar surrounding most of the article (item 1 in Figure 1.1) is easily dealt with by either using the wiki markup as a starting point or by taking advantage of the common XHTML structure shared by the articles. Advertisements and spam are exceedingly rare on Wikipedia and it seems like there is little point in attempting to identify them. Forms (i.e. pages that accepts input from the user) are non-existent in the main articles. Duplicate content mainly exists in the form of templates, and the problems posed by them are different than those of identifying quoted text. Our approach used in the second phase of cleaning is inspired by one of the systems (namely NCLEANER) participating in this shared task.

1.2.2 KrdWrd

KrdWrd (Steger and Stemle, 2009) is a system for annotating web pages and building web corpora. One of its components is a Firefox plugin that makes it possible to annotate pages in a browser where they appear as they normally do. Its annotation guidelines[4] are based on those used in CleanEval (Steger and Stemle, 2009; Pomikálek, 2011) and uses the terms "good" and "bad" for text that should be included in a corpus and not respectively. The requirements for clean text is stricter in the KrdWrd guidelines than they are in the CleanEval guidelines, as the following

[4] `https://krdwrd.org/manual/html/node6.html`

types of boilerplate is added in addition to what was already considered boilerplate in CleanEval: Incomplete sentences, text in foreign languages, text containing file names and other "non-words" and enumerations (unless it is a complete sentence). Meta-information is not explicitly mentioned, but it likely falls under the general description of boilerplate: "Generally speaking, boilerplate is everything that [...] could be left out without changing the general content of the page."

The annotation guidelines also include a third category, "uncertain", for text that do not match the criteria for "good" or "bad" text. Annotators are instructed to mark captions, headings, labels and so on as uncertain.

By only including full sentences, what constitutes clean text according to KrdWrd resembles our notion of relevant linguistic content. The most striking difference is that headings are not considered clean, another difference is that "non-words" and text in foreign languages are to be annotated as "bad". Strictly speaking, no content is removed from the web pages during the annotation, but there no way of knowing why an element is marked as "uncertain" (headings are always marked as uncertain, which effectively leaves them un-annotated). In our view it is better to include these elements and try to label them appropriately (our approach is described in Chapter 3).

The Canola corpus, which we will refer to later in Section 4.3.2, is one of the resources created using the KrdWrd system. This corpus is consists of 216 web pages that have each been annotated by 5-12 annotators (Pomikálek, 2011, p. 38).

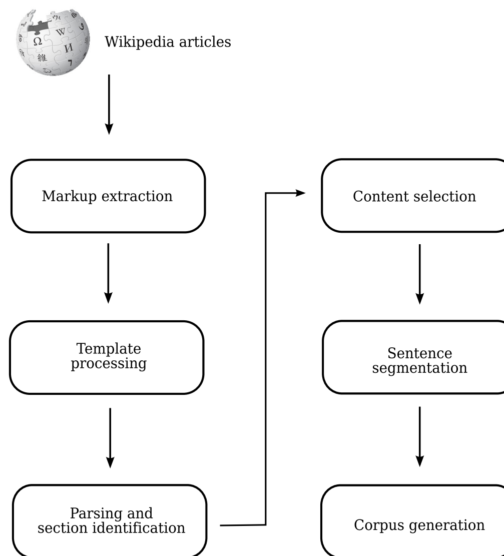
1.2.3 L3S-GN1

The L3S-GN1 data set (Kohlschütter et al., 2010)[5] is a collection of manually annotated news articles collected from Google News. It was created to serve as a gold standard for evaluating page cleaning approaches. The annotations in L3S-GN1 marks the page content as either: "...headline, fulltext, supplemental (text which belongs to the article but is not fulltext, such as image captions etc.), user comments, related content (links to other articles etc.). Unselected text is regarded not content (boilerplate)." (Kohlschütter et al., 2010, p. 443).

The "related content" class would be considered noise for our purposes and would also be considered as boilerplate by the CleanEval and KrdWrd annotation guidelines. The "supplemental" class seems to fill a similar role as the "uncertain" class used by KrdWrd with the exception of headings that

[5]Available at: `http://www.L3S.de/~kohlschuetter/boilerplate`

Figure 1.2: Overview of our system



are annotated as "headline". Of the sets we are aware of, "user comments" is unique for L3S-GN1.

1.3 Thesis Overview

The next chapter discusses the structure of Wikipedia and gives a brief introduction to wiki markup and its significance for downstream processes. Furthermore it surveys earlier efforts in extracting content from Wikipedia and some tools capable of processing wiki markup.

The rest of this thesis closely mirrors the structure of our system that is sketched in Figure 1.2. Chapter 3 describes the first three stages starting with how we process a database snapshot of Wikipedia in order to extract the wiki markup. In the "Template processing" stage (Section 3.3) we discard a substantial amount of noise by selectively expanding templates, which is the first phase of cleaning. Some templates contain information that is valuable to downstream users, for instance those that are used to mark up dates or inline citations, and we explicitly include the presence of those in the corpus. The final wiki markup is then parsed and each article is then split at the section level in the "Parsing and section identification" stage (Section 3.4).

The second phase of cleaning is done by classifying content as either relevant linguistic content or noise and is described in Chapter 4. This is the "Content selection" stage in the sketch. We make use of machine

learning in our approach in order to recognise types of noise we have not encountered ourselves. After this stage all remaining content should ideally be clean.

Before we create a corpus all text is split into sentences in the "Sentence segmentation" stage described in Chapter 5. Here we test a range of tools and perform experiments with different approaches for harnessing markup. We perform our tests on WeScience, a corpus consisting of 100 Wikipedia articles with gold standard sentence segmentation.

Finally in Chapter 6 we describe the end product of running our system on a Wikipedia snapshot: A large high-quality corpus with little noise and where the more interesting wiki markup directives are annotated with Grammatical Markup Language (GML), a low-verbosity language that is designed to cover the linguistically relevant directives from several other popular markup languages like wiki markup, HTML and LaTeX.

1.4 Summary of Main Results

Below is a quick summary of the main results in this thesis:

- We have developed a method for creating high-quality corpora from collections of user generated content. We also make an implementation and a large corpora based on a database snapshot of Wikipedia available.

- Our approach to templates, a class of markup directives that have program-like properties, makes it feasible to apply a relatively low number of hand-written rules on a large majority of template inclusions.

- We furthermore describe an effective, both in throughput and in classification accuracy, method of identifying relevant linguistic content.

- We present a survey of sentence segmenting tools and offer several methods of using the markup elements (or layout information) as a way of increasing their performance.

Chapter 2

Background and Motivation

This chapter provides relevant background for the project, including a run down of the organization of pages on Wikipedia and an introduction to wiki markup. Some of the markup elements in the article source do have some linguistic significance and will be included in the corpus. We will (in Section 2.2) examine some of the previous efforts on either directly using Wikipedia as a resource for NLP research or creating a community resource from it. Section 2.3 discusses existing tools for processing wiki markup, these are both academic and commercial.

2.1 Format and Structure of Wikipedia

Wikipedia runs on Mediawiki[1], this software was originally developed for Wikipedia but is now used by several other wikis. Pages are written in a markup language, called wiki markup, and are converted to XHTML when presented to a visitor. It is possible to download compressed snapshots, also called "dumps", that contain the wiki markup for articles and templates from `http://dumps.wikimedia.org/`. In order to compare our work with WeScience and WikiWoods (see Section 2.2.2), we have chosen to use the same snapshot[2] from 2008 that was used in the creation of those corpora. The approach described here can be used on newer Wikipedia dumps as well as dumps from other Mediawiki wikis.

Pages are organised by type into namespaces, the notation used to refer to pages in a namepsace is "Namespace:Page", but the namespace part can be left out when referring to pages in the main namespace (also called "the nameless namespace"). When we use the term "article" we refer to the

[1] `http://www.mediawiki.org`

[2] Available at `http://moin.delph-in.net/WikiWoods`

Table 2.1: Pages per namespace in our snapshot

Namespace	Function	Pages	Redirects
Nameless	The main namespace, this is were the articles are placed.	2,496,177	2,964,714
File	All images, sound files and other uploaded files have a page describing their licence, revision history and so on. These pages and the uploaded files live in the "File" namespace.	825,955	63
Category	A category is a list of pages, a page is included in a category by linking to it.	389,980	201
Wikipedia	Pages concerning editing policies, coordination efforts, various projects etc.	308,669	57,070
Template	Templates	144,933	29,244
Portal	Portals are collections of links to various articles with a common theme.	56,036	3,886
MediaWiki	Files that are used in the user interface (css files, links in the navigation frame and so on).	901	16
Help	Pages explaining how to use and contribute to Wikipedia.	193	240
Book	This namespace is for collecting articles into books that can be exported or printed using the Collection extension.	0	1
User	Personal pages for contributors.	0	0
Special	Pages with special functions, like user lists and create a permanent link to the current article. This is a virtual namespace, meaning that the pages here are generated on the fly.	n/a	n/a
Media	Direct links to the files in the "File" namespace, this is virtual namespace.	n/a	n/a

pages in this namespace. Table 2.1 shows the distribution of regular pages and redirects for each of the namespaces used by Wikipedia. Some of the namespaces are not included in the dump at all, others are very scarcely populated. Each namespace has a corresponding "talk" namespace (e.g. "Talk", "Template talk", "Help talk" and so on) that is used for collaboration and discussion among the page authors. This is done by editing a page in the talk namespace with the same name as the page being discussed, for instance discussion on "Albert Einstein" takes place in "Talk:Albert Einstein". Pages in the talk namespaces are not included in the dump.

Since our goal is to mine content from the articles our main interest is the pages in the main namespace. But in order to correctly parse the wiki markup for those pages we also need access to the templates, and these pages reside in the "Template" namespace.

2.1.1 Short Primer on Wiki Markup

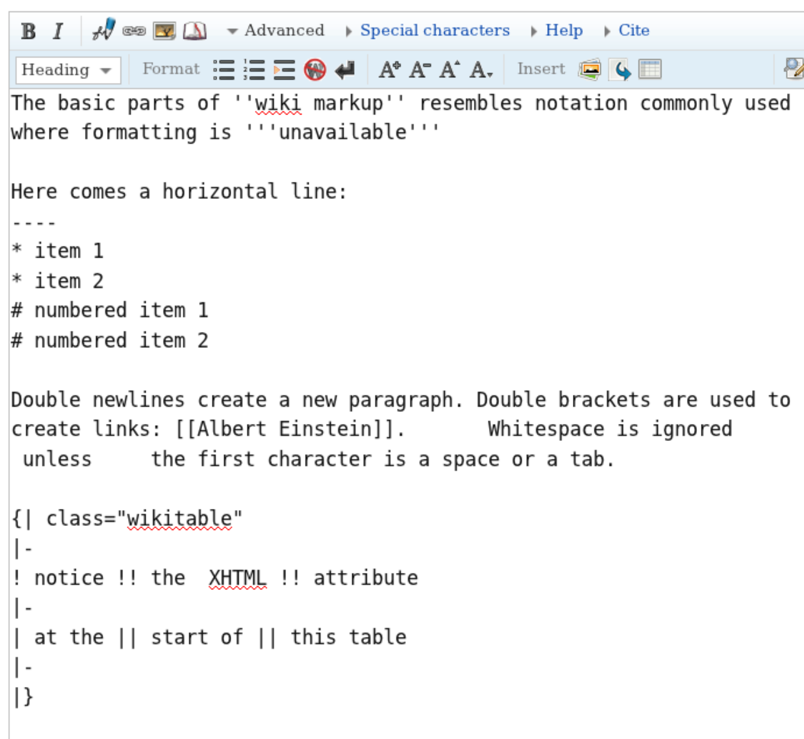


Figure 2.1: Wikipedia's page editor

Hypothetically, a page made up of nothing but plain text will in most cases appear as one would expect when processed by the Mediawiki engine,

The basic parts of *wiki markup* resembles notation commonly used where formatting is **unavailable**

Here comes a horizontal line:

- item 1
- item 2

1. numbered item 1
2. numbered item 2

Double newlines create a new paragraph. Double brackets are used to create links: Albert Einstein. Whitespace is ignored

```
unless     the first character is a space or a tab.
```

notice	the XHTML	attribute
at the	start of	this table

Figure 2.2: A simple page in Wikipedia

but most articles make use of at least some markup to indicate topical structure, create links and basic formatting. The most basic markup directives resemble conventions sometimes used in place of formatting in plain text. For instance lines that start with * or # are displayed as list elements, while ---- will create a horizontal line. Two consecutive newlines are treated as a paragraph break. Plain URLs are converted into links and inter-article links are specified by double square brackets. Figure 2.1 shows a sample of the wiki markup and Figure 2.2 how it appears when rendered.

As one would expect from a system that outputs XHTML, the characters less than (<) and greater than (>) are usually replaced by XML entities (< and >). The exception are when they are part of a certain subset of permitted XHTML tags. These tags pass trough the parser unchanged and can be assigned attributes like this .... Some elements, for instance tables (<table>...</table> or {|...|}), can be created by both regular XHTML tags and wiki markup. Wikipedia uses the extension "Math" that adds support for rendering mathematical formulas by enclosing LaTeX statements in <math> tags.

Starting a page with #REDIRECT [[Albert Einstein]] creates a redirect to the page enclosed in square brackets. When a redirect is accessed normally the content of the target article is shown instead with a small notice on the top of the informing the viewer that they have been redirected.

This directive is often used in order to let searches and links using different naming conventions lead to the same article (for instance both "Einstein" and "A. Einstein"[3] are both redirect to "Albert Einstein"). Is is also used to create short convenient aliases for pages with long titles, like in our dump where `Template:Harvtxt` redirects to `Template:Harvard citation text`. Redirects function in all namespaces and they are honoured during template expansion.

Mediawiki is extremely permissive and robust when it parses the markup and we are not aware of any way to construct wiki source that does not render (although the rendered result might differ from what was intended). Wikipedia maintains a markup guide at `http://en.wikipedia.org/wiki/Help:Wiki_markup`, but the language is not formally defined[4].

2.1.2 Linguistically Relevant Markup

Some markup directive contain information that can be useful for several NLP tasks. In-text links from Wikipedia was exploited in Nothman (2008) for named entity recognition, Section 2.2.1 takes a closer look at this effort. Spitkovsky et al. (2010) took advantage of anchors and text styles when training an unsupervised dependency parser and got a marked increase in parsing accuracy. It seems reasonable to believe that using selected markup elements as parsing constraints will also be useful for parsing in general. If we look beyond the markup elements used in Nothman (2008) and Spitkovsky et al. (2010), wiki source often contain templates that can be used to identify text as dates, in text citations, in a foreign language and so on. Markup also plays an important role, in what is often considered a pre-processing task, sentence segmentation. We discribe our approach to segmenting marked up text in Chapter 5.

In our system we attempt to enable such approaches by retaining markup elements that we consider might be of use to downstream processors. Each type of markup element is treated in one of the following ways:

1. Included as a GML tag: These are the elements that usually have some semantic meaning, they generally fall into tree sub-categories: Text styles (bold, italic, etc), logical tags (list, abbreviation, paragraph, ...) and various link types. This process is referred to as ersatzing.

[3] `http://en.wikipedia.org/wiki/Einstein` and `http://en.wikipedia.org/wiki/A._Einstein` As of Sept. 16. 2012.

[4] There are work being done to create a specification, see `http://www.mediawiki.org/wiki/Markup_spec`.

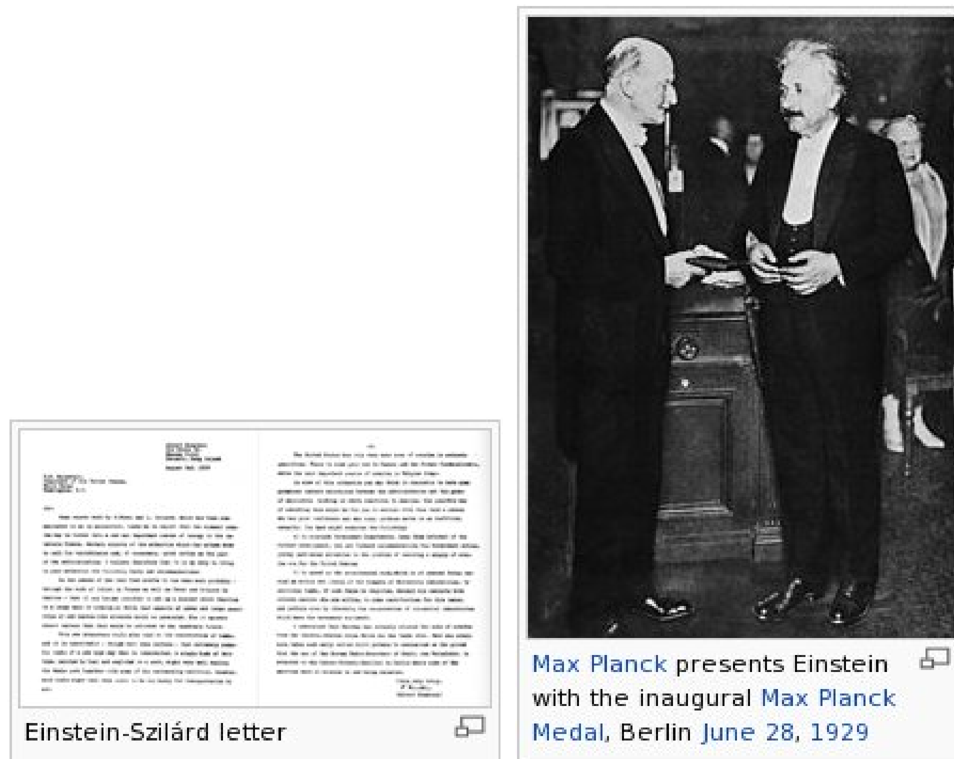


Figure 2.3: Pictures from the article "Albert Einstein"

2. Replaced by an empty GML tag: Markup elements that are replaced are those that can be used as a sentence constituent while not having any content that is immediately useful. In-line images are the only elements that are handled this way.

3. Only its content is included: This option is used for elements like the <span> and <center> tags, that often contain linguistic content, but the semantics of this content is not affected by their presence.

4. Neither the element nor its content are included: Some markup elements never contain any relevant linguistic content and are considered noise, category links[5], horizontal lines, certain templates etc. We also remove certain elements that some times are clean, namely image captions and tables. The reasoning behind this is explained below.

[5]Used to include an article in a category, these are rendered in a box at the end of the page

For some markup elements it is not immediately clear how they should be treated, for instance both image captions and table cells occasionally contain relevant linguistic content. The caption for the left image in figure 2.3 is very short and does not qualify as relevant linguistic content, while the caption for the other picture is a complete sentence containing information not present in the main text of the article. Images and their captions are removed at the cost of losing a few relevant phrases, as keeping them would introduce unwanted content. The exception to this are in-line images that are replaced with an ersatz token, as they often are a constituent in a sentence and removing them would leave behind ill-formed sentences. In a somewhat similar vein the content of table cells can be fairly long spans of natural language or something that is obviously non-linguistic (dates, numbers, etc). Unfortunately, the meaning of a phrase in a table cell is often highly dependent on row and column headings and without it those phrases will have little value for a semantic parser. A complete list of the different syntactic elements and how they are treated is in Appendix A.

2.1.3 Templates

Templates are pages that can be included in other pages, a common example is "Fact" that contains the phrase "citation needed", it is used to draw attention to statements that should cite a source of some kind. There are other templates that are more advanced, like the many "information box" templates, that take several parameters and expand into a table like box of labels and short descriptions.

Including, or "expanding", a template in wiki source text is done by placing the template name between double curly brackets, as for example `{{SomeTemplate}}`. This will cause Mediawiki to insert the page "SomeTemplate" in the current article, if "SomeTemplate" is a redirect the page it redirects to will be inserted in its stead. Templates reside in the "Template" namespace, but it's possible to include any page by explicitly specifying the namespace it is in, as for example `{{User:Username/SomePage}}` or for the main (nameless) namespace: `{{:SomeArticle}}`. If used in an article, the last example would result in a article including another, a technique sometimes used to maintain long "List of"[6] articles.

Template inclusion happens before most of the other markup is processed. When discussing this subset of wiki markup it is useful to give it a name in order to be able to easily contrast it with the "regular wiki

[6] E.g. "List of asteroids/1-1000"

Listing 2.1: The definition for "Flag"

```
{{country data {{{1|}}}
| country flag2
| name = {{{name|{{{1|}}}}}}
| variant = {{{variant|{{{2|}}}}}}
| size = {{{size|}}}
}}<noinclude>{{documentation}}</noinclude>
```

markup", we will in this thesis refer to it as "template markup". Even though, as we will become apparent in Section 2.1.3.1, it is the part of wiki markup that is furthest from what one usually considers markup.

Since template markup is evaluated before the regular parsing takes place, it is possible for templates to expand into whole or partial markup elements. For example the frequently used "End" expands into `|}`, the directive for end of a table. Removing this template, as is often done in naive approaches to wiki markup processing, will mean that everything up to the next section header is interpreted to be in a single table cell, something that has the potential to cause large portions of relevant linguistic content to be discarded. There are also several templates that insert the markup for table start (`{|`), the removal of those would make the table body appear as regular text containing several vertical bars (`|`). Something that will introduce a lot of noise into the corpus, a problem that can be seen in the WikiWoods corpus (see section 2.3.3).

2.1.3.1 More Advanced Templates

While many templates simply insert static text into an article, Mediawiki offers several features that can be used to create more intricate templates. The most important of these features are: argument passing, evaluation of mathematical expressions and conditional execution.

Listing 2.1 shows the wiki markup for "Flag", a template that accepts both positional and named parameters. Each of them are represented as numbers or strings inside of triple curly braces, where their default value follows directly after the horizontal bar, i.e. `{{{1|}}}` refers to the first positional argument and its default value is the empty string. Had it not been set, the default value would have been "undefined" which means that the variable would have been expanded to the literal "{{{1}}}".

Listing 2.2: Snippet from the source code for the template “Fb r”

```
<!--
if r equals "null"                                                    -->|{{#ifeq:{{{r}}}|null<!--
   then set background color
      -->|bgcolor=#CCCCCC{{!}}{{Unicode| }}<!--
   else if gf is not empty (set background color)                     -->|{{#if:{{{gf|}}}<!--
           if home team wins
              -->|{{#ifexpr:{{{gf}}}-{{{ga}}}>0<!--
              then set background color                               -->|bgcolor=#DFE7FF{{!}}<!--
              else if away team wins
                 -->|{{#ifexpr:{{{gf}}}-{{{ga}}}<0<!--
                      then set background color                       -->|bgcolor=#FFDFDF{{!}}<!--
                      else (it's a draw) set background color         -->|bgcolor=#FFFFFF{{!}}<!--
                   endif                                              -->}}<!--
           endif                                                      -->}}<!--
        if ma in not null                                             -->{{#if:{{{ma|}}}<!--
           then "[[ma|"                                               -->|{{open
              wikilink}}{{{ma}}}{{!}}<!--
        endif                                                         -->}}<!--
        if gf is not null                                             -->{{#if:{{{gf|}}}<!--
           then gf-ga                                                 -->|{{{gf}}}&ndash;{{{ga}}}<!--
           else if ma is not null                                     -->|{{#if:{{{ma|}}}<!--
                   then "a"                                           -->|a<!--
                   else " "                                           -->|{{Unicode| }}<!--
                endif                                                 -->}}<!--
        endif                                                         -->}}<!--
        if ma is not null                                             -->{{#if:{{{ma|}}}<!--
           then "]]"                                                  -->|{{close wikilink}}<!--
        endif                                                         -->}}<!--
endif                                                                 -->}}
```

Listing 2.3: The expansion of `{{Flag|China}}`

```
{{country data China
| country flag2
| name = China
| variant =
| size =
}}
```

Anything between `<noinclude>` and `</noinclude>` is only interpreted when the template is viewed directly, a facility often used to document the usage of the template (usually by including the template "Documentation"). If invoked like this `{{Flag|China}}` this template expands into the text shown in listing 2.3.

The result of this expansion is, as shown in Listing 2.3, contained between double curly brackets. Creating the markup for including another template: "Country data China". It is one of the many templates that are not intended for direct inclusion in articles, their usage resembles that of subroutines in programs: They allow for code re-use (both "Flag" and "Flagicon" include "Country data ..." templates) and makes it possible to split problems into smaller and more manageable chunks. The final result, via a few other templates, is the markup for a tiny image of the Chinese flag followed by a link to the article "People's Republic of China".

Recursive template inclusions are limited by Mediawiki in that a template can only include itself once[7], either directly or via other templates, and that expansion stops when the call-stack reaches a certain depth [8].

Wikipedia uses the extension "ParserFunctions" that makes flow control and mathematical operations available to the Wikipedia authors. These are evaluated before the regular markup and can be used to conditionally expand templates. Listing 2.2 shows most of "Fb r", a template that is used to create a stylised cell in a table ("FB r" stands for "football result"). This template is somewhat atypical as the author has indented and commented their code. The HTML-style comments start at the right side of each line and end about two thirds across the page from the left on the next. The code in this example shows the resemblance between Mediawiki templates and other programming languages. One thing lacking in ParserFunctions that one would expect from a fully fledged programming language is loops,

---

[7] `http://en.wikipedia.org/w/index.php?title=Help:Template&oldid=478928626` As of Feb 26. 2012.

[8] `http://www.mediawiki.org/w/index.php?title=Manual:\protect\T1\textdollarwgMaxTemplateDepth&oldid=196093` As of Feb 26. 2012.

a limitation that seems like a sensible move considering that anyone can edit Wikipedia pages and it would be desirable if it were impossible to create pages that would never finish rendering.

2.2 Previous Work

There has been much effort both on using Wikipedia as a resource as well as making its content more accessible for researchers. The creation of the WeScience and WikiWoods corpora are probably the projects that resembles our the most. This section will provide an overview of some of the earlier uses of Wikipedia in NLP.

2.2.1 Wikipedia for Named Entity Recognition

How links between Wikipedia articles might be used as a tool in named entity recognition was examined by Nothman (2008). One example he gives is the sentence "Holden is an Australian automaker based in Port Melbourne, Victoria" where each of the proper nouns link to an article that, when classified, can be used to identify the type of entity they refer to (Nothman, 2008, p. 33-34).

While his objective is different from ours, there are some similarities in the general approach: extract the article markup from a Wikipedia dump, parse it in order to extract the linguistic content, and detect sentence boundaries. This makes it worthwhile to take a look at his methods. He examined several processing systems, including WikiXML and mwlib (both discussed below in Section 2.2.3 and 2.3.2), before deciding to use mwlib as a basis for creating a parser. This choice seems to be motivated by the fact that mwlib offers access to the processed wiki markup as a parse tree (Nothman, 2008, p. 40).

For sentence segmentation he used the Punkt (Kiss and Strunk, 2006) implementation included in the "Natural Language Toolkit"[9]. Its performance is informally summed up as "generally produced reasonable sentence breaks", but it is also noted that it struggled some times when facing abbreviations directly followed by a word that it deemed to be a frequent sentence starter (Nothman, 2008, p. 41-42). We did test NLTK's Punkt implementation and several other sentence boundary detectors on the WeScience corpus to determine which segmenter to use in our system. The results of these experiments are presented in Chapter 5.

[9] `http://www.nltk.org/`

Listing 2.4: One sentence from WeScience

```
[10011140] |* '''Recursion''' or '''iteration''': A
   [[recursive algorithm]] is one that invokes (makes
   reference to) itself repeatedly until a certain
   condition matches, which is a method common to
   [[functional programming]].
```

2.2.2 WikiWoods and WeScience

WeScience[10] and WikiWoods[11] are corpora created from a Wikipedia dump from July 2008. WeScience consists of 100 articles in the NLP domain with gold standard sentence segmentation (Ytrestøl et al., 2009). WikiWoods is a larger corpus that contains around 1.3 million articles (Flickinger et al., 2010). Corpus Clean (described in section 2.3.3) was used in the creation of both of them. Corpus Clean is not capable of fully parsing wiki markup and as a consequence of this WikiWoods contains an unnecessary amount of noise. It makes some steps to remove dirty sections, but both of these corpora have a relatively high concentration of sections with little relevant linguistic content.

They both have the same line-based format with one sentence per line, with some of the original wiki markup preserved. Listing 2.4 shows one line from WeScience. Enclosed in square brackets is a unique sentence identifier, where the last digit was initially set to zero in order to make room for manual adjustments of the sentence segmentation. The sentence itself starts after the vertical bar and continues for the rest of the line (Ytrestøl et al., 2009; Flickinger et al., 2010).

2.2.3 WikiXML

WikiXML is a collection of Wikipedia articles in XML format created by The University of Amsterdam. Both the collection itself and the conversion software are available at their website[12]. The conversion software consists of a modified version of Mediawiki and a post-processing script in Perl. The files in the collection are in valid XML that resembles the XHTML generated by Mediawiki and are viewable with a web browser. The results of template inclusions are for the most part marked as such. Template parameters are also included when the software was able to extract them. The spans

[10] http://moin.delph-in.net/WeScience
[11] http://moin.delph-in.net/WikiWoods
[12] http://ilps.science.uva.nl/WikiXML/

that are included from templates are, a bit awkwardly, represented by pairs of self-closing tags: `<wx:template id="wx_t1" ... />` marks the beginning of an expansion, and `<wx:templateend start="wx_t1"/>` marks the end. Where the `id` and `start` parameters are used to match them up. Since templates can expand into anything, this is probably one of the best ways of including them and still generating valid XML. We feel that templates are one of the reasons that XML is not a good choice for representing content that originated as wiki markup, something that is discussed in more detail later in Section 6.1.

The table of contents, edit-links and the frame surrounding the article are removed, no other steps are taken to remove boilerplate and other noise. This stands in stark contrast to our approach. Our aim is to produce a corpus that can be used "out of the box" and in order to achieve this we have a much more aggressive approach when it comes to article cleaning.

Parsing well formed XML is a less daunting task than parsing Mediawiki markup and WikiXML could probably fit in as a first step in a wiki-text processing pipeline, but we decided against using WikiXML this way since we felt that the increased complexity by having an extra step would outweigh the convenience of parsing XML. The fact that the preservation of template inclusions is not reliable, as stated on their web page, also spoke against building upon WikiXML.

Using WikiXML was a candidate approach for processing a dump in Nothman (2008), but it was considered to be "excessively slow" (Nothman, 2008, p. 38) and mwlib was used instead.

2.3 Tools for processing Wikipedia Dumps

We will in this section review some of the existing tools for processing Wikipedia snapshots and parsing wiki markup. Wwlib and Corpus Clean have already been mentioned above, but will be examined more closely. The Wikimedia Foundation maintains a list of several wiki parsers[13], but unfortunately most of the tools listed there are either limited in scope or too immature to be of immediate use for us.

[13] http://www.mediawiki.org/wiki/Alternative_parsers

2.3.1 DumpHTML

DumpHTML is a tool that was used by the Wikimedia Foundation to create static HTML dumps of Wikipedia. The tool itself is maintained [14] even though static dumps are no longer offered.

DumpHTML uses the Mediawiki rendering engine, something that makes it dependent on a properly configured database back-end. It outputs XHTML files that are similar to what is presented when visiting Wikipedia. This means that the XHTML pages include several elements that are not present in the wiki markup, such as navigational bars, copyright notices, table of contents etc. Just like the regular Mediawiki, DumpHTML expands templates transparently and there is generally no way to tell which content is generated by a template inclusion from the content in the article source. This is undesirable because some of the wiki markup can be helpful in the semantic analysis of the text and it increases the difficulty of distinguishing genuine authored text from phrases inserted by templates.

Modifying Mediawiki so that it doesn't generate navigation bars, etc is fairly straightforward and WikiXML shows that it's possible to retain some information of template usage. Had we decided on an approach using dumpHTML, we could probably take advantage of some of the code in WikiXML. We will discuss why we chose not to build on the Mediawiki engine in our system below.

2.3.2 Mwlib

Mwlib is the result of a collaboration between the Wikimedia Foundation and Pediapress.[15] It is used in the conjunction with the extension "Collection" that adds the functionality to create collections of articles that can either be ordered as a printed book or exported into several document formats. It was successfully used to process a Wikipedia dump in Nothman (2008). Mwlib is implemented in Python and C and is actively maintained.

In the early stages of this work we examined the available tools for manipulating Mediawiki markup (late 2011) and at that time mwlib was one of two parsers on The Wikimedia Foundation's list of parsers that offered access to the syntax tree. The other was the perl module "Perl Wikimedia Toolkit", but it was not considered as it was labelled as "Little functional". At the time of writing a few new tools have been added, a cursory glance gives the impression that these are still a bit immature (low percentage of

[14] http://svn.wikimedia.org/viewvc/mediawiki/trunk/extensions/DumpHTML/ As of Sept 14. 2012 the latest commit was about 6 weeks old.

[15] http://wikimediafoundation.org/wiki/Wikis_Go_Printable

Listing 2.5: Excerpt from WikiWoods

```
[1000898100080] |===Round 1===
[1000898100090] ||- bgcolor="#CCCCFF" | '''Home team''' |
    ...
[1000898100100] |===Round 2===
[1000898100110] ||- bgcolor="#CCCCFF" | '''Home team''' |
    ...
[1000898100120] |===Round 3===
[1000898100130] ||- bgcolor="#CCCCFF" | '''Home team''' |
    ...
[1000898100140] |===Round 4===
```

successful parses or missing support for some of the popular extensions) to be of practical use for us.

Support for working with dumps was removed from mwlib with the release of version 0.14. A third party module providing this functionality is available at `https://github.com/doozan/mwlib.cdb`.

2.3.3 Corpus Clean

Corpus Clean was used in the creation of the WeScience and WikiWoods corpora (Ytrestøl et al., 2009; Flickinger et al., 2010). It consists of several Python scripts and an open-source tool, Tokenizer[16], that is used to detect sentence boundaries. It operates on files with wiki markup.

Not all of the content in an article makes it through the pipeline, as Corpus Clean makes an effort to only include clean text in the resulting corpus. This cleaning is done by removing some types of markup elements like tables, images and, with a few exceptions, templates. In addition to this a heuristic approach is used to remove sections that contain little grammatical text (Ytrestøl, 2009).

Since it operates on the textual level by repeatedly matching and replacing strings in its input with regular expressions, it is unable to perform proper template handling. Corpus Clean has a white-list of six templates that are kept in the text but the default action is to remove them. While templates are a frequent source of noise, uncritically removing them will introduce errors into the wiki markup that it is hard to recover from. This is due to their ability to expand into any syntactic elements (as mentioned in section 2.1.3). For instance, searching for `|` in WikiWoods yields several examples of stray table cells that Corpus Clean has failed to remove because

[16]Available at`http://www.cis.uni-muenchen.de/~wastl/misc/`

of this. One example of this is shown in Listing 2.5, this fragment is taken from the article "1936 VFL season" as it appears in WikiWoods. Corpus Clean could not detect the tables since it did not examine the content of the surrounding templates "Start box" and "End box" that expand into table begin (`{|`) and table end (`|}`).

Tokenizer is not markup-aware, so it struggles when tasked with segmenting text that was written with a specific layout in mind (i.e. containing markup). It often fails to insert sentence breaks when encountering formatting that would normally cause a human reader to interpret a span of text as a separate sentence. Ytrestøl mentions lists as one type of markup that could cause this problem. Corpus Clean's solution to this is to forcefully insert a sentence break in such cases. Other sources of errors were missing or unusual punctuation and confusing headers and captions with the main text (Ytrestøl, 2009, p. 8-9). We have a somewhat similar approach, but since we have access to a parse tree we have the opportunity to take greater advantage of the markup when finding sentence boundaries.

Chapter 3

Article Extraction and Parsing

The steps “Markup extraction”, “Template processing” and “Parsing and section identification” sketched in Figure 3.1 will be described in more detail in this chapter. These stages obtain article source and process it in a way that leaves us with each article section represented as a tree. A data structure that it is convenient to work with in and that will be used in the remaining stages. A large part of this chapter will be dedicated to the choices we have made concerning templates and the reasoning behind them. Templates are both a source for noise and for linguistically significant markup, something that we have taken advantage of in our system.

Figure 3.1: Overview of our system

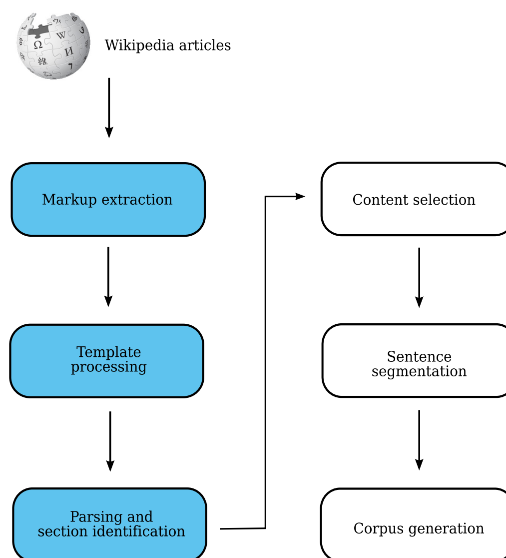


Table 3.1: Comparison of dumpHTML and mwlib

	dumpHTML	mwlib
time needed to build a “text extractor”	approx. 12 hours	approx. 5 hours
documentation	excellent	minimal
code readability	poor	good
implementation language	php	python
parsing approach	series of string manipulations	building a parse tree
template capabilities	good	good
parsing correctness	almost guaranteed	good

3.1 Choosing a Wiki Parser

As discussed in Section 2.3, a number of packages for processing wiki markup are available. Seeing as markup processing was bound to be a central part of our system we wanted to make sure we picked the right tool. After doing an initial survey of the available tools we narrowed the candidates down to dumpHTML (outlined in Section 2.3.1) and mwlib (outlined in Section 2.3.2). Both of these have properties that we considered desirable:

- DumpHTML uses the Mediawiki rendering engine, which is the closest thing there is to a formal specification of wiki markup.

- mwlib builds a parse tree, where it seemed reasonable to us that having the parse articles represented in a familiar data structure would simplify further processing.

We were however unsure how we should weigh these characteristics. In the spirit of our general approach of gathering data when in doubt, we performed an experiment of extracting all printable text from 1,000 articles. The primary objective was to gauge the amount of effort it would take to build this simple program using each of the two candidates.

It took substantially more time and effort to extract content from articles by modifying dumpHTML than it took when using mwlib. The two main reasons for this were: (a) that dumpHTML depends on an operational installation of Mediawiki, including the same extensions used by Wikipedia.

Though the procedure for setting this up is well documented, [1] carrying it out still took some time. Seeing as we had set out to build a system that could be reused by others, having it depend on Mediawiki and a number of extensions was not desirable. And (b) even though Mediawiki's inner workings are thoroughly documented[2] we had some difficulties making sense of the source code. Most of the parsing is done by loading the article source into memory as a string and gradually rewriting it until it is a valid XHTML document.

Working with mwlib was a much more straightforward. It builds an abstract syntax tree while parsing, so extracting the text from an article was simply a matter of supplying the article name to the parser and traversing the resulting tree. The documentation[3] is mostly geared towards system administrators, so the main focus is on installing the library and configuring it so that it functions with an existing mediawiki setup. The code itself is generally readable.

Table 3.1 gives an overview of dumpHTML and mwlib and summarises this experiment. The result of which was that we decided to use mwlib to process wiki markup.

3.2 Markup Extraction

Recall from the outline of our system sketched in Figure 1.2 that the first step is "Markup Extraction". Our system can read markup from a Wikipedia dump or from plain files with wiki markup. When reading from plain files the article name is either inferred from the file name, for instance when reading from a file named "Albert Einstein.mw" the article name will set to "Albert Einstein". Or it can be set by adding article tags at the start of each file, like this: `<article>Albert Einstein</article>`. This is not a proper wiki markup directive and it is used solely for convenience. Article tags are removed before any further processing of the markup. Article tags are also used by Corpus Clean to indicate the start of a new article and its title.

Extraction from a snapshot is done by first creating a "Constant database" with the "mw-buildcdb" utility bundled with mwlib, this database then functions as a back-end for mwlib (the usual choice for back-end is a live Mediawiki instance with the Collection extension installed).

[1] http://www.mediawiki.org/wiki/Manual:Importing_XML_dumps

[2] http://www.mediawiki.org/wiki/Manual:Code

[3] http://mwlib.readthedocs.org/en/latest/index.html

3.3 Templates

Template expansion is one of the first steps in parsing wiki markup and plays an important part in the removal of noise. As described in section 2.1.3 some templates have program-like features and they can also expand into arbitrary strings. This means that the result of an expansion can include things like partial wiki markup directives and partial sentences, and that simply removing templates will lead to ill-formed markup and loss of content. A result of this is that proper interpretation of templates is essential for our system.

Since mwlib is capable of expanding templates (although we had to make some adjustments to its template system, described in Sections 3.3.3 and 3.3.5 below) we can do proper inclusions when the need arises. We make use of this flexibility to clean articles by removing templates that we know introduce noise and by enriching the corpus by making the presence of those that might be useful for further linguistic processing explicit.

3.3.1 As a Source of Noise

Template inclusions create a lot of noise, as they are commonly used to insert boilerplate both inside running text (e.g. "Citation needed") and as separate block elements (for instance the boxes shown in Figure 1.1). In order to create a corpus that is as clean as possible we try to remove as many of the noise-introducing templates as possible. However care must be taken so that we do not remove templates that expand into partial markup elements or parts of relevant linguistic content. As can be seen by examining the WikiWoods corpus (for an example see Listing 2.5), being too aggressive in removing templates is counter-productive as it will introduce noise.

3.3.2 As Cues for Downstream Programs

Some templates can not only expand into parts of natural text, but their presence could also aid further linguistic analysis. Take for instance the template "Lang". The documentation gives the following example[4]:

> She said: "{{lang|fr|Je suis française.}}".

This template is used to indicate the language of a span of text. If it is expanded then the task of figuring out that "Je suis française" is not English is passed on to downstream language processing systems. Not only

[4]italics has been removed for clarity.

is it desirable to keep such templates for their added value, but removing them fully will often leave behind gaps in the original text. This has the potential to transform relevant linguistic content into noise, for instance the above example would end up as “She said: ”.

3.3.3 Our Strategy

Being too defensive and expanding all templates will waste a good opportunity to remove a substantial amount of noise as the content of templates like “Fact”[5] will be inserted into the articles. On the other hand, removing too many of them will not only cost us useful content, but will actually introduce noise (as was the case with Corpus Clean). When our system encounters a template, it will take one of three actions:

- It can be expanded as normal, and we will call this action **expand**.
- It can be removed completely, and we will call this action **remove**.
- The invocation can be left in the corpus alongside its expanded form, we will call this action **keep**. The example used above would look like this in the markup language we use in our corpus (GML) is: `⌊x¦Je suis française.¦Lang¦fr¦Je suis française.¦x⌋`.

We created a modified version of the template sub-system in mwlib that is capable of these three actions and we manually inspected the most commonly used templates in order to create a list of rules, where each template were assigned one of the above actions. When our system encounters a template not covered by this list it expands it as normal, since that is the safest approach for dealing with unknown templates. The look-up operation is aware of redirects (described in section 2.1.1) between templates, so a rule defined for “Harvard citation text”[6] will be honoured for all templates that redirects to it, for instance “Harvtxt”. A positive side effect of this is that inclusions of redirects templates are normalised to the name of their target, so downstream applications do not need to have any knowledge about inter-template redirects.

When deciding how each template should be treated we used the following guidelines:

1. Can the presence of this template be helpful for further natural language processing? **keep**.

[5]Expands into the text “[Citation needed]”
[6]Creates an in-text Harvard style citation.

2. Do we know that this template creates noise and that it can be removed without breaking markup? **remove**.

3. Do we know that this template creates noise and that it can be removed without altering sentence structure? **remove**.

4. Does the template reside in the main namespace (i.e is it an article)? **remove**.

5. If none of the above, **expand**.

The reason for not removing inclusions of pages in the main namespace (point 4) is that we are already including all articles. Expanding those template inclusions would lead to repetitions of the same content in the corpus.

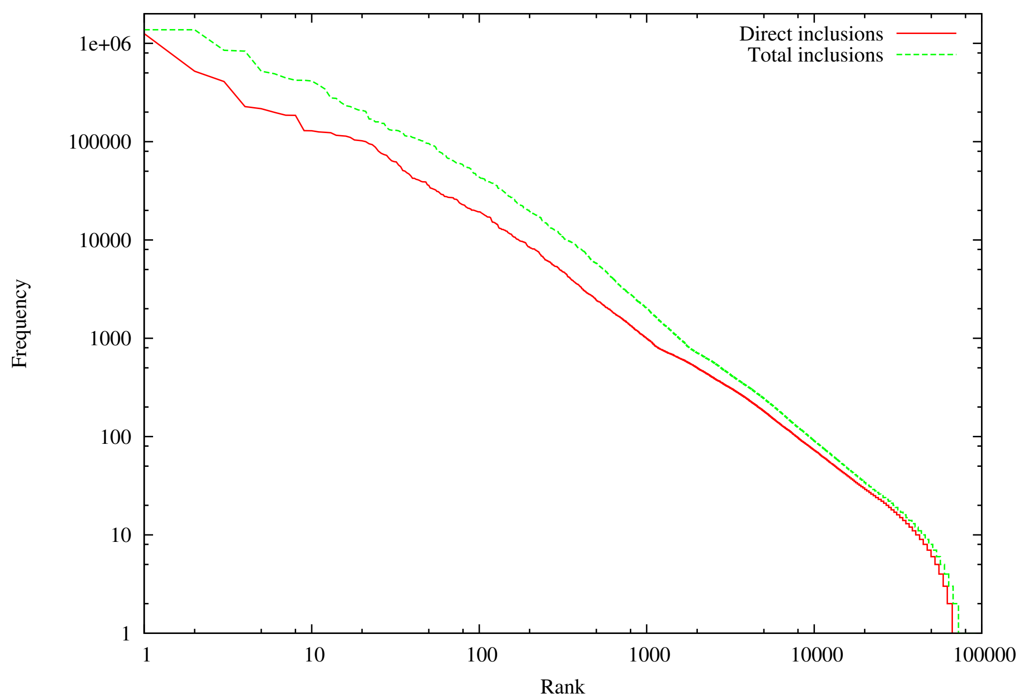
3.3.4 Finding the Most Used Templates

The efficiency of the above strategy depends on the coverage of our set of rules. In the dump we are using, the `Template` namespace contains over 140,000 pages (as shown in Table 2.1). Since there is no way of determining the appropriate action for a template without doing a manual inspection, we could not create rules for all of them. The next best thing seemed to be to find the most frequently used templates and write rules for those.

As described in section 2.1.3, where `{{Flag|China}}` expanded into the markup for including `{{Country data China}}`, templates can be invoked indirectly. We wanted to capture both modes of inclusion in our examination. Using the template handling mechanism in mwlib as a starting point we counted all inclusions in our snapshot. The direct inclusions are found by feeding the article source to the first step of mwlib's template facility, where this creates a tree where each node is either a template invocation (a `Template`-object containing the template name and a list of arguments) or a text string. In order to get the count of both direct and indirect inclusions, we passed this tree into the next stage of the template system (called "flattening" in mwlib terminology). The flattening stage recursively expands any templates in the tree and creates a list of tokens, where the beginning and end of each template expansion is represented. Since templates can expand into the markup for including other templates, the flattening is repeated as necessary. See Listing 3.1 for an example of the tokens created during template expansion.

One special case at this point is the wiki `<ref>` element that is used to create end-notes. Template inclusions inside of these were ignored as they

Figure 3.2: Distribution of templates



are mostly used to create references, often by using one of the many cite templates, that we would remove anyway. Comparing the counts with and without the contents of `<ref>` showed that the treatment of this tag only marginally affected the top-100 lists.

This inspection showed that there are more than 90,000 templates that are included at least once from articles (that is pages from the main namespace) in our dump. Figure 3.2 shows the frequency of template inclusions after ranking by the number of direct or total number of inclusions plotted in log-scale. Notice how relatively few templates stand for a large number of inclusions. The most frequently used template is expanded over a million times while the 100th most included is used less than 20,000 times.

The steep increase in the number of inclusions meant that writing rules for a few of the highest ranking templates would cover a large amount of the actual inclusions. It was however unclear if the best approach would be to order by direct or total inclusions. When sorting by total inclusions some templates rose towards the top that could otherwise have gone by unnoticed. Take for instance the template "Ambox" that creates a message box. There are several templates that functions as wrappers and do little more than

Table 3.2: Most included templates

Direct	Total	Name	Description	Action
1256248	1375733	Flagicon	A small flag	remove
520927	520982	Reflist	List of references	remove
409021	415170	Cite web	Reference to a web site	remove
227473	227578	Fact	"[Citation needed]"	remove
216303	217974	Cite journal	Reference to a journal	remove
198974	207888	Convert	Converts between units	keep
186172	275796	Mp	`{{Mp|x|n}}` shows x to nth power	expand
185066	203211	Cite book	Creates a reference	remove
129606	421673	Flag	Small flag next to a link to a nation.	expand
129244	130316	Succession box	Expands into table cells	expand

including it with predefined parameters (one of them is "Refimprove" shown at the top of Figure 1.1). Any rule written for "Ambox" will indirectly affect all of these wrappers and increase the coverage of our rules. On the other hand, not all of the templates that showed up when sorting this way were that interesting. There are for instance many templates with names like "Country data Norway", "Country data Sweden" and so on that get their vast majority of inclusions via "Flag" and "Flagicon", templates that occur frequently enough to warrant their own rules. The high ranking of such templates pushes other potentially interesting templates down the list.

In order to get the best of both worlds we examined the 100 most frequently included templates using both ways of counting, this resulted in a list of 155 templates that we each assigned with one of the three rules: "keep", "remove" or "expand". Table 3.2 shows first the ten entries in this list and is included in Appendix B in its entirety.

We remove the most frequently included template, "Flagicon", completely as it expands into a small image of a flag and we do not anticipate this being used as a grammatical constituent. "Flag" resembles "Flagicon" in some ways and expands into a icon of a flag followed by a link to a nation. We expand this template as it could be used as part of a sentence. "Cite web", "Cite journal" and "Cite book" all create bibliography entries, which we do not consider to be relevant linguistic content and are removed. The template "Convert" is used to change from one measuring unit to another

(e.g. `{{convert|3.21|kg|lb}}` expands into `3.21 kilograms (7.1 lb)`). We keep this template since its presence does signify that the expanded text is a "quantity of something".

3.3.4.1 Naming Conventions

One of the templates we examined was "Asbox", which expands into a notice saying "This article is a stub. You can help Wikipedia by expanding it." It is used in short articles and has two uses: it encourages people to add more content to the article, and it adds the article to one or more stub categories. Within Mediawiki's browsable category system, stub categories are used to group articles that need attention. This template is only explicitly included 10 times, but its total number of inclusions are a bit higher than 59,000. When doing some initial surveys of our snapshot we found 500,000 articles where the source code consists of less than 1,000 characters. When including articles of up to 2,000 characters in their markup gave a total 1,200,000.

Casually browsing Wikipedia gives the impression that most of the short articles have such a notice, but "Asbox" was not included nearly enough to be the source of more than a fraction of those. Even if we take into consideration that many of the short articles were disambiguation pages or not consider stubs for some other reason.

A closer examination of the template counts revealed several thousand templates that had a similar function and followed the naming convention "<something>stub" ("England-cricket-bio-stub", "Astronomy-stub", ...), their combined number of direct inclusions was 1,268,560 which is slightly more than the current number one "Flagicon".

In order to find other groups of templates that followed a naming convention we looked for frequently used sub-strings in template names. Each sub-string occurring in template names was assigned a score that equalled the sum of direct inclusions of all matching templates, and the resulting list was then sorted by this score. The number of expansions for each template was only counted once even if it could be matched with several sub-strings. This was to avoid getting the results cluttered with very similar groups that had little variation in the names they matched (e.g "stub", "-stub", "b-stub", ...). The 100 most expanded templates were not included as they had already been accounted for.

The first attempt had no special treatment for white-space characters and other frequent word boundaries, as we wanted to see find naming conventions even if they spanned several words. Unfortunately this gave a high score to many uninteresting sub-strings that matched several unrelated templates, for instance: "s of ". Splitting names on word boundaries gave

Table 3.3: Template naming conventions

Inclusions	Matching regexp	Description	Action
1268560	stub$	Adds a notice saying that this article is a stub	remove
581244	^infobox	Creates an information box	remove
68355	^lang	Indicates that a span if text is of a given language	keep
54706	county	A navigation box for American counties	remove
28958	^cite	Creates a citation	remove
28316	^pbb	PBB = Protein Box Bot, information boxes about proteins	remove
26971	line$	Information box about a subway/rail/bus line	remove
19432	^auto	Displays a quantity of something in a given measuring unit	keep
17240	expand$	Same function as /stub$/	remove
12375	communes$	Information boxes for French communes	remove
11715	^IPA	Displays text in IPA notation	keep
9640	^politics	Navigation boxes for politics related articles	remove

a much more interesting list of candidates. Out of the 50 highest scoring patterns the naming conventions shown in Table 3.3 were assigned a rule other than expand. The complete list is in Appendix B.

3.3.4.2 Final Additions to the Rules

Finally, the templates that were kept by Corpus Clean were added to the list. Most of these were marked as “keep” except “IPA notice”, which creates a box with a short introduction to IPA, and “IAST” and “IAST1” that both expand into “Lang” that is kept. The inclusion of “IPA notice” appears to be a result of using a regular expression to match the other IPA templates without being aware of that specific template. Corpus Clean also kept templates that created in-line Harvard style citations, most of these have a redirect with a shorter name for convenience for instance “Harvtxt” redirects to “Harvard citation text”. Since Corpus Clean only looks for the shorter

name of the redirect, all inclusions using the full name are lost.

There have been a few additions to the rules-set as we have uncovered other templates that should not be expanded. The grand total of direct template inclusions from the main namespace in our dump is in excess of 14 million, where about 10 million of these are of templates that are covered by our rules. As the coverage of the rules increases the gain from adding new rules becomes smaller. This is because the majority of the remaining templates are rarely used.

3.3.5 Modifications to Mwlib's Template System

As mentioned in Section 3.3.3 we had to do some modifications to mwlib's template system in order to carry out the actions "keep", "remove" and "expand". We also had to do some tweaking in order to increase the speed of the template expansions. In order to explain these modifications we first need to take a look at how templates are expanded by mwlib. The template sub-system has two stages; The first is a pass over the article source creates a parse tree where each node is either a `Template`-object or a unicode string. `Template`s extend the python built-in `Tuple` where the template name is the first element and any parameters are in a list as the second element.

The second stage is to "flatten" this tree, that is to traverse it and include the content of all the templates. Any template inclusions done by these templates are parsed and flattened themselves. This results in a list of tokens, named "marks" internally in mwlib.

Most of these tokens are regular strings containing wiki markup. The rest indicate the beginning and end of template expansions (signalled by `mark_start` and `mark_end` tokens) or positions where it might be necessary to insert extra newlines to ensure that statements that need to be at the beginning of a line are interpreted correctly (signalled by `mark_maybe_newline`).

Listing 3.1 shows the first few marks when expanding `{{Flag|China}}`. By examining the start and end-marks we can see that "China" was expanded into "Country data china" that in turn expanded into an image and a link via the expansion of "Country flag2". The `<mark 'dummy'>`'s are used to pad the list during flattening and have no impact on the expanded markup. The final wiki markup is then created from this list by concatenating the string-tokens and selectively inserting newlines where there is a `mark_maybe_newline`.

The actions "remove" and "expand" are easily implemented by making a pass over the list of marks. Whenever there is a `mark_start` for a template that should be removed, it and all marks up to and including the

Listing 3.1: Marks created during the expansion of `{{Flag|China}}`

```
<mark_start "u'flag'">
<mark_start "u'country data China'">
<mark_argument "u' country flag2\\n'">
u'<span class="flagicon">[[Image:'
u"Flag of the People's Republic of China.svg"
u'|'
<mark_maybe_newline 'maybe_newline'>
u'22x20px'
<mark 'dummy'>
u'|'
u'border'
u' |'
u"Flag of the People's Republic of China"
u']] </span>[['
u"People's Republic of China"
u'|'
u'China'
u']]'
<mark_end "u'country flag2'">
<mark_end "u'country data China'">
<mark_end "u'flag'">
```

corresponding `mark_end` are removed. No action is necessary for templates that should be expanded normally.

Implementing "keep" required a bit more work since we need both the expanded form of the template and the invocation parameters. Unfortunately for us the parameters are normally discarded during flattening. We solved this by implementing a alternate flatten procedure for the `Template` class that inserts `mark_arguments` into the list of marks. The expansion of `{{Flag|China}}` with this modification is shown in listing 3.2 (see Listing 2.1 for its definition). Notice that the arguments sent to "Country data" and "Flagicon2" preserved in the stream of tokens. When we iterate over the marks we emit strings with the expanded form, template name, and any parameters for templates where the rules say "keep". Any template inclusions embedded in the arguments are expanded as normal.

The second change we made to the template system was motivated by performance issues. When we first started counting template inclusions it took almost five days to expand all the inclusions from the articles in the dump. Even though we were aware that the finished system would run several concurrent processes, expanding templates would only be one of the several sub-tasks that it must do. While looking for ways to increase the

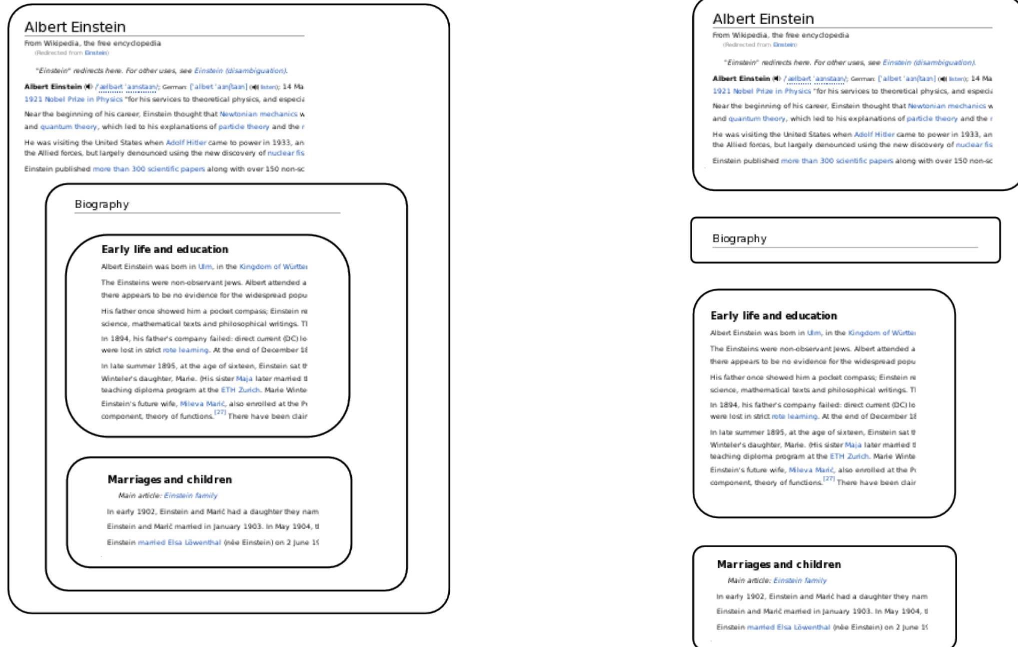
Listing 3.2: Marks created during the expansion of `{{Flag|China}}` with the arguments preserved.

```
<mark_start "u'flag'">
<mark_argument "u'China'">
<mark_start "u'country data China'">
<mark_argument "u' country flag2\\n'">
<mark_argument "u' name = China\\n'">
<mark_argument "u' variant = \\n'">
<mark_argument "u' size = \\n'">
<mark_start "u'country flag2'">
<mark_argument 'u" alias = People\'s Republic of China\\n"'>
<mark_argument "u' shortname alias = China\\n'">
<mark_argument 'u" flag alias = Flag of the People\'s
   Republic of China.svg\\n"'>

[snip...]

<mark_argument "u' variant = \\n\\n'">
u'<span class="flagicon">[[Image:'
u"Flag of the People's Republic of China.svg"
u'|'
<mark_maybe_newline 'maybe_newline'>
u'22x20px'
<mark 'dummy'>
u'|'
u'border'
u' |'
u"Flag of the People's Republic of China"
u']] </span>[['
u"People's Republic of China"
u'|'
u'China'
u']]'
<mark_end "u'country flag2'">
<mark_end "u'country data China'">
<mark_end "u'flag'">
```

Figure 3.3: Article structure before and after section identification



performance we found out that mwlib caches templates and that this cache is purged every time an article is parsed. By preserving the content of the cache between the pages, the running time went down to about two days at the cost of increased memory consumption.

3.4 Section Identification

When the template handling is done the resulting template-free wiki markup is parsed, and the parse tree is destructively split into sections.

Mwlib considers a section to be a heading and everything up to the next heading of the same or lower level. We have opted to view the article structure in a different way as sketched in Figure 3.3: A section consists of a heading and all that follows up to the next heading, no matter its level. This means that all sections are in separate parse trees and that they never contain other sections, something that simplifies the section classification step as the classifier does not need to have any notion of subsections. It is worth keeping in mind that no information is lost by this transformation as the original structure is easily recoverable by traversing the list of sections backwards. If the next section that has the same level as the current one is its sibling, those with a higher level is the child of a

sibling and those with a lower level is a parent. We employ this technique in the last stage of our system in order to completely leave out sections with little relevant linguistic content on the condition that they do not have any children carrying "clean" text. We include the heading of these sections in order to preserve the structure of the article.

We will repeatedly need to produce string representations of the section trees. This is in order to communicate with third-party programs and, off course, when creating the final representation of our corpus. This is done by traversing the syntax trees and emitting text nodes and markup directives when appropriate. Since this process is done several times, we name the four different ways we can treat each markup element in order to facilitate discussions on this topic in the following chapters.

When creating a string representations of a sections, we must choose between the following actions detailed in Section 2.1.2:

keep Insert some markup into the string so the existence of this node is visible. Any children are dealt with depending on their type.

remove Do not insert any markup for this node. Children are treated according to their type.

purge Do not include this node or its children in the string.

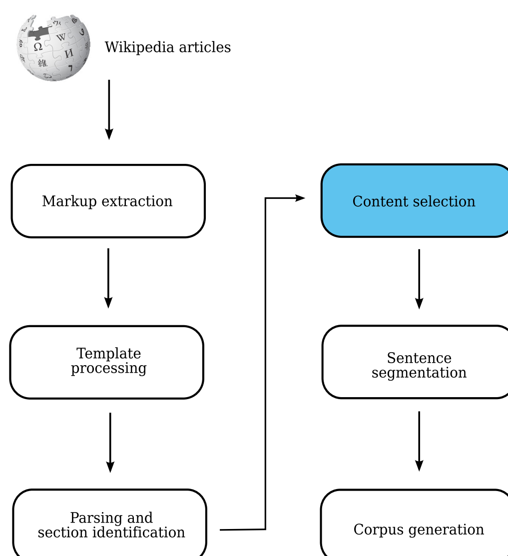
ersatz Replace this node with a token string and do not include any children.

Chapter 4

Content Selection

This Chapter explains the "Content selection" stage in our system of which an overview is sketched in Figure 4.1. At this point articles have been parsed and the resulting parse trees have been split into smaller section-wise branches. During the parsing we removed a substantial amount of noise by selectively discarding certain templates (as detailed in Section 3.3). Unfortunately, there are still plenty of noise left that we can not identify by only looking at the markup. This noise is not generated by known templates or other specific markup directives. Examples of this type of noise are bibliographies, result listings form sporting events, footnotes and so on.

Figure 4.1: Overview of our system



We will examine some earlier efforts on noise removal which with the exception of Corpus Clean all operate on ordinary web pages. While there are large overlaps between extracting clean text from Wikipedia articles and from web pages there are also some fundamental differences (as mentioned in Section 1.2): Some types of content are practically non-existent on Wikipedia like advertisements and web-forms, while copyright notices, navigation bars and so on are not an issue since we base our approach on processing the article source. We do, on the other hand, have to deal with noise in the form of bibliographies, various listings, and so on, which are probably more common in Wikipedia than on most web pages.

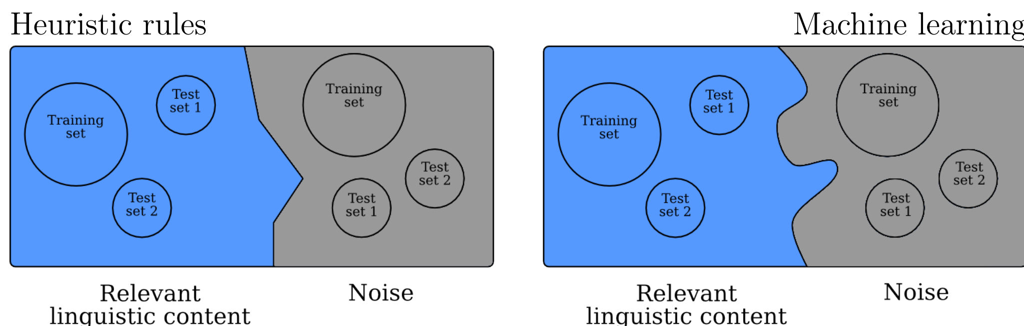
4.1 Hand-Crafted Rules vs. Machine Learning

There are several ways to classify content as clean or dirty; Corpus Clean used a heuristic approach where it discarded a section, and in some cases the rest of the article if it had a heading that was usually used for sections with little relevant linguistic content (Ytrestøl, 2009, p. 9). A downside of using hand-written rules is that these rules will be skewed towards the instances of noisy text seen by the author. The page cleaning tool jusText, which will be reviewed in Section 4.3.2, uses more general criteria when detecting noise. These criteria include, for instance, link density and the frequency of function words. Rules of this kind might be less vulnerable to this effect as they depend on the more generalised properties, but they still depend on some assumptions about the characteristics of clean and dirty content.

Conversely, an approach using machine learning might be able to identify "dirty" sections that would slip past a set of explicit rules, possibly by using characteristics of the two classes that do not stand out enough for a human to identify them. The criteria determining if something is relevant linguistic content or not are determined by training on existing samples, that could very well be collected using an heuristic methods, and the classifier would hopefully then be able to generalise from this data.

Figure 4.2 illustrates schematically how a classifier using machine learning might have a more fine grained view of what is relevant linguistic content or not than a classifier using hand-written rules. The separation of clean (coloured blue) and dirty (coloured gray) content follows a few straight lines for the heuristic classifier as it depends on a limited, hand-picked, set of characteristics for both classes. The classifier to the right

Figure 4.2: Separation of clean and dirty sections using heuristic rules and machine learning



that relies on machine learning has a more granular separation of clean and dirty content, as shown in the figure by the curved border between the two classes.

4.2 Background: N-gram Models

N-gram models play an important role in the remainder of this chapter as they have been used in previous efforts on page cleaning (namely NCLEANER that is reviewed in Section 4.3.1) and are used in our system (described in Section 4.4). Therefore, we will spend a few paragraphs detailing them here. An N-gram model gives an estimate of the probability of events by examining it and a finite number of previous events.

Before we can describe n-gram models we need to say a few words about conditional probabilities. The probability (P) of event A given event B is:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{4.1}$$

Where $P(A \cap B)$ is probability of both A and B taking place. Multiplying both sides of Equation 4.1 with the denominator ($P(B)$) gives us the following:

$$P(A|B)P(B) = P(A \cap B) \tag{4.2}$$

We can then use the chain rule to find the probability for multiple events:

$$P(A \cap B \cap C) = P(A)P(B|A)P(C|A \cap B) \tag{4.3}$$

The chain rule can be restated as follows, where W is the sequence of events of length k:

$$P(W) = \prod_{i=1}^{k} P(w_i|w_1^{i-1}) \tag{4.4}$$

A Markov assumption is that the probability of an event only depends on a finite number of previous events. We can use this to estimate the probability of a sequence in the following way:

$$P(W) \approx \prod_{i=1}^{n} P(w_i|w_{i-(n-1)}^{i-1}) \tag{4.5}$$

N-gram models use the Markov assumption to estimate the probability of an event by examining the probability of the event given the $n-1$ events before it. The term order is used to refer to the size of n. Unigram models (i.e. $n = 1$) do not use the history when finding the probability and consider all events to be independent (Jurafsky and Martin, 2009, p. 120-125). An n-gram is a sequence of events of length n.

The events can in principle be anything, but at least when used in NLP they are often words or characters. The probabilities are determined by training the model, and this is in turn done by counting the n-grams of length n and $n-1$ in a data set. The probability of an n-gram, say "dogs chase cats", is calculated by dividing the number of times it occurred in the training data by the count of the shorter n-gram "dogs chase":

$$P(cats|dogs\ chase) = \frac{C(dogs\ chase\ cats)}{C(dogs\ chase)} \tag{4.6}$$

The probability for whole sequences is estimated using Equation 4.5.

From the description above a sequence that was not in the training data (for instance "dogs chase buses") would get an estimate of zero as $C(dogs\ chase\ buses) = 0$. This is undesirable as this is something that is bound to happen due to the great variation of natural language. The way this is handled is by shifting some of the probability mass from the known n-grams over to the unseen ones. This is a technique called smoothing. We discuss smoothing algorithms in Section 4.5.2.2 as we test several of these techniques in our experiments.

4.2.1 Perplexity

Measuring the perplexity gives us an impression of how well a language model describes a string. The perplexity (PP) of a string of words

Table 4.1: Top-five performers in the "Text-only" part of CleanEval

System	Score
Victor (Marek et al., 2007)	84.1
GenieKnows (Weizheng and Abou-Assaleh, 2007)	83.4
Kimatu (Saralegi and Leturia, 2007)	83.4
Hofmann and Weerkamp (2007)	83.0
StupidOS[a] (Evert, 2007)	82.9
Htmlcleaner (Girardi, 2007)	82.5
Chakrabarti et al. (2007)	80.9
FIASCO (Bauer et al., 2007)	73.5
Conradie	60.2

[a] NCLEANER participated under the name of StupidOS.

$W = \{w_1 w_2 ... w_k\}$ with a model of order n is calculated as follows:

$$PP(W) = \sqrt[k]{\prod_{i=1}^{k} \frac{1}{P(w_i | w_{i-(n-1)}^{i-1})}} \tag{4.7}$$

That is the probability of string normalised by the number of words in it. Since the probability is in the denominator a lower perplexity means a better prediction of the set (Jurafsky and Martin, 2009, p. 129-131).

4.3 Previous Work

There have been several initiatives for extracting linguistic content from on-line sources. The CleanEval shared task (outlined in Section 1.2) that was held in 2007 prompted the creation of several tools that employed a range of different techniques. One of the participants, NCLEANER, will be detailed below as our approach borrows heavily from it. We will also examine Corpus Clean, which was the tool used to create the corpora WikiWoods and WeScience from a Wikipedia snapshot. Finally we will look at JusText, that is a newer algorithm for page cleaning (Pomikálek, 2011) which outperforms all of the participants in CleanEval.

4.3.1 NCLEANER

NCLEANER (Evert, 2008) was one of the participating systems in the CleanEval shared task. It has an uncomplicated architecture but, as shown

in Table 4.1, it was still competitive with the top performers on the "Text-only" part of the competition. The scoring metric used was modified Levenshtein edit distance divided by file length (Baroni et al., 2008). The average score in the "Text-only" part was 79.3. NCLEANER is available for download.[1]

NCLEANER classifies text by running it through two character-level n-gram models, one that is trained on unwanted text (the "dirty" model) and one that is trained on the desired content (the "clean" model). The probability scores received from these models are then compared and the class is determined by observing which model gives the highest probability. The granularity of this classification is paragraphs, headings and list items.

A few preproccesing steps are performed before the segments are fed to the language models: The first step removes a set of HTML elements (images, comments and JavaScripts) and replaces line break tags (`<br>`) with paragraph tags (`<p>`). Markers that help with the identification of headings and list items in step three are inserted. The second step converts the HTML into plain text using Lynx (a text-only web browser). In the third step all non-Ascii characters are replaced with the tilde character (~) and the page is split into the three segment types mentioned above. Segments that can be identified as noise by heuristic measures are removed. One such heuristic is "blocks where multiple fields are separated by vertical bars" (Evert, 2008, p. 3490). The remaining segments are then classified by comparing the probability scores given by the two language models.

The units that are classified (paragraph, header and list item) are the same as those used in the document restructuring task in CleanEval. While it might be sensible to classify entire paragraphs, it seems a bit strange to classify typically short elements as headings and list items as clean or dirty without examining their context. Especially when considering that the scores given to short segments could be greatly affected by the presence of a single word that is often seen in boilerplate, like "home" or "click". One could imagine cases where this would lead to a heading being classified as noise while a paragraph directly following it is classified as clean.

4.3.2 JusText

JusText (Pomikálek, 2011) is an algorithm that takes a heuristic approach to cleaning web pages. A Python implementation[2] and a live demonstration page[3] are available on-line. It makes two passes over the page, where the

[1] http://webascorpus.sourceforge.net/

[2] http://code.google.com/p/justext/

[3] http://nlp.fi.muni.cz/projects/justext/

first pass classifies segments when they are long enough and the heuristics for doing so are fulfilled with a large enough margin. The remaining segments are marked as "short" or "near-good", and these are classified in the second pass by examining their neighbours. The intuition is that clean segments are usually surrounded by other clean segments and vice versa.

Segmentation is done by splitting the page on block elements, and the features used in the first pass are length, link density and density of function words. As with NCLEANER, the segments that gets classified can be quite small (e.g. headings, table cells and so on), but jusText attacks this problem by deferring the classification of those to the second pass which does not examine their content at all.

JusText was developed after CleanEval and outperforms all of the participants on the data set used in the shared task. It was also tested alongside some of the tools that participated in CleanEval (including NCLEANER) on the Canola corpus (Steger and Stemle, 2009) and L3S-GN1 data set (both are described in Section 1.2). JusText is among the best performers on all of these data sets (Pomikálek, 2011, p. 47–51), but it is difficult to pick the best system from these tests due to different tools coming out on top for different data sets and scoring metrics.

There are also some differences in what content is considered clean in the different sets. In L3S-GN1 all content "annotated as one of the five main content classes" (Pomikálek, 2011, p. 50) was considered clean. Presumably this includes the class "related content" that is used for links to other articles as often seen on on-line news sites, this is something that would be marked as boilerplate in the other data sets. Additionally, the KrdWrd guidelines call for annotating all content that is not comprised of complete English sentences as noise, which is a requirement that is not in the CleanEval guidelines.

4.3.3 Corpus Clean

The tool used in the creation of the WeScience and WikiWoods corpora (Ytrestøl et al., 2009) (described in Section 2.2.2) has a simple heuristic approach to section classification. As mentioned in Section 2.3.3, Corpus Clean removes article sections after only examining their heading. Like jusText, it uses the intuition that noise is usually surrounded by more noise, and whenever it encounters one of the following section headings, it discards the rest of the article:

- related web sites
- external links

- bibliography
- footnotes

Furthermore, the following section headings trigger the removal of the rest of the article as long as the same heading is not repeated later in the text:

- see also
- notes
- references
- sources

These rules aims to leverage the article conventions in use on Wikipedia. However, they are tuned to the articles examined during their creation. While we use machine learning in our approach to noise reduction, we used heuristic rules not too different from those used by Corpus Clean for collecting test and training data. The content of these data sets are limited by our abilities to formulate good heuristics, but this is mitigated by our classifiers ability to generalise. The process of collection training data is described in Section 4.5.1 below and our classifier is described in Section 4.4.2.

4.4 Our Approach

Our approach is inspired by NCLEANER (outlined in Section 4.3.1). Even though it was not the winner of CleanEval it was competitive with the other participants (as seen in Table 4.1) while at the same time not being particularly resource intensive. In addition, its general architecture is not tied to any specific format. Basing our approach on a pair of n-gram models also leaves the door open for extending our classifier by using its output as a feature for a classifier using another type of machine learning.

We use two character-level n-gram models for classifying content, where the classification is done by comparing the probabilities given to the text by the models. If the model trained on relevant linguistic content gives the highest likelihood, the text is kept. If the model trained on noise gives the highest probability, the content is discarded.

Even though NCLEANER is publicly available we do not use it as the classifier in our system, but have instead opted to used the SRI Language Modelling Toolkit (SRILM) (Stolcke et al., 2011). Since we already have parse trees of the articles the effort of doing the preproccesing

ourselves is about the same as creating HTML files that can be classified by NCLEANER. Doing the preprocessing ourselves also has the advantage of freeing us from the limited set of markup elements used by NCLEANER (`<h>`, `<l>` and `<p>`) by allowing us to use any format we see fit for during classification. We believe this is important for performance as the density and composition of markup is often noticeably different for "clean" and "dirty" text. We also expect that SRILM (implemented in C++) outperforms NCLEANER (implemented in Perl) in terms of speed, which is important considering that we are operating on a snapshot of the complete Wikipedia.

We have chosen to view article sections as singular units for the purpose of classification as this granularity has several advantages. One of them is that we will not be classifying very short spans of text that can be vulnerable to the presence of atypical properties that can lead to misclassification. For instance headings without any context. This could also lead to some some odd situations where a heading is classified as dirty while the rest of the section is classified as clean.

Another advantage is that we do not run the risk of splitting sentences, take for instance this sentence that starts in a regular paragraph and ends as a list from the article "Cons":

> ... Such a list can be created in three steps:
>
> 1. Cons 3 onto nil, the empty list
> 2. Cons 2 onto the result
> 3. Cons 1 onto the result

Removing the three steps would remove a large part of the last sentence of the preceding paragraph, making it ungrammatical and effectively turning it into noise.

There was also the practical matter of obtaining training and test data (detailed in Section 4.5.1), which we generated by matching sections in our dump to a number of criteria. It turned out that finding good heuristics for "clean" and "dirty" sections was not as trivial as it may sound, and we had to run two iterations before we were satisfied with the results. Specifying criteria for smaller, sub-section, chunks of clean and dirty content would probably have been harder.

There is however one disadvantage to classifying such large chunks of content: That even though most sections are made up of a single type of content, there will inevitably be some sections that contain both relevant linguistic content and noise. We will return to these "mixed" sections

Listing 4.1: A section after preproccesing.

```
<h3>C++ Library </h3>
<p><ul><li><NamedURL>Crypto++</NamedURL> A comprehensive
   C++ semi-public-domain implementation of encryption and
   hash algorithms. FIPS validated</li>
<li><NamedURL>Chris Lomont's version of AES under the zlib
   License</NamedURL></li></ul></p>
```

in Section 4.6 where we take another look at what constitutes relevant linguistic content.

4.4.1 Preproccesing

Recall from Chapter 3 that at this stage articles are internally represented as individual parse trees in our system and that these trees are never nested. In addition, all template inclusions are either removed, expanded or made explicit (as detailed in Section 3.3.3). The templates that we keep are now represented as regular text nodes in a parse tree.

Before they can be classified a string representation must be created for each section. This corresponds to the preproccesing done by NCLEANER. It differs from the approach taken by Evert (2008) in that we have chosen a much more verbose format for these strings, as shown in the example in Listing 4.1. We have done so because we believe that the classification step would benefit from the presence of markup, as this does in a way make certain features like markup density and link count available to our classifier. We build these strings so that they contain the markup that has some linguistic relevance, while elements that introduce noise are silently ignored (tables, purely navigational elements and so on, see Appendix A for a complete list).

With a few minor exceptions this string contains the same information as the final GML representation that will be used in the corpus. The reason for not using GML was that it only existed on the conceptual stage when we constructed this part of the system. But the conceptional separation between these strings and the final representation is intentional as some page elements might be useful cues to the classifier even if we do not want to include them in the corpus. One example might be "category links" that are usually found near the end of an article and are used to include an article in a category or images that do not follow the normal text flow.

Any sections where the string representation only contains a heading are not evaluated by the classifier, but their inclusion in the corpus is

instead determined by whether or not they have any clean sub-sections (as described in Section 3.4). The string could be empty because there were two consecutive headings in the articles source, something that is not uncommon on Wikipedia. Other reasons could be that all content was removed during the template expansion or that all remaining content was in the form of markup elements that are not included in the classification.

4.4.2 The Classifier

As mentioned above, we use SRILM to build and apply our n-gram models. SRILM supports a range of smoothing techniques, which alongside the n-gram order and quality of the training data have a major impact on the performance. We experimented with different configurations (discussed in Section 4.5), before settling in the one we use in our system.

SRILM has the ability to run as a server, this is convenient for us as we can start two servers (one for each model) and connect to these from any number of concurrent processes. The network protocol is text-based and fairly straightforward: The client sends an n-gram where words are separated by spaces and the server responds with the logarithm of the probability. It is up to the client to multiply (or sum since $log(x)+log(y)$ is equivalent to $x \times y$) the replies in order to get the probability for the whole string.

In order to operate on the character level we reformat the string by inserting spaces between all non white-space characters. It is then converted to Ascii by passing it to Python's string.encode('ascii', 'backslashreplace'), which replaces non-Ascii characters with a backslash followed by a hexadecimal number (e.g. the character ø is replaced by \xf8), in order to prevent any encoding issues. A sentence-start (<s>) and sentence-end (</s>) token is then put at the beginning and end of each line respectively.

A telnet session with a server loaded with the dirty model, where we get the probability for the top level heading "Notes" (<s> < h 1 > N o t e s < / h 1 > </s> after preproccesing and formatting) is shown in Listing 4.2. Line one shows the salutation sent by the server upon connecting. Following on line two is the first ngram from our heading, which gets a probability close to one ($10^{-0.0021} \approx 0.9952$). This is an effect of the format used during the classification where almost all lines starts with a tag, and hence the first bigram is almost always <s> <. Also, notice how some of the n-grams receive extremely low probabilities (line 6 and line 26). There is also a smaller dip for the n-gram h 1 > N (line 10) compared to that of its neighbours, as was not in the training set and the server had to back off

Listing 4.2: Telnet session with a SRILM server.

```
probserver ready
<s> <
-0.00210638
<s> < h
-0.627294
<s> < h 1
-5.48345
< h 1 >
-0.09691
h 1 > N
-2.84527
1 > N o
-0.546763
> N o t
-0.397856
N o t e
-0.181237
o t e s
-0.386462
t e s <
-0.333785
e s < /
-0.00273824
s < / h
-0.263875
< / h 1
-5.48344
/ h 1 >
-0.653212
h 1 > </s>
-0.39794
```

until it finally found a matching bi-gram.

Some care must be taken when communicating with the server in order to get a decent performance, while passing too few n-grams at once leads to a lot of transmission overhead, passing too many before reading the replies leads to a deadlock as the server blocks when its output buffer is full. Implementation-wise this meant that we had to do some tuning in order to find the optimal amount of data to pass to `socket.sendall()`, which is the function used to send data over a network connection.

4.5 Finding the Optimal Configuration

The performance of our classifier depends on several factors, where the most important are:

The training data The amount and quality of the training data, and as we mentioned in Section 4.4 even the format of text being classified plays an role. We did two iterations where we collected training data from our Wikipedia dump.

N-gram order If the order is to small, the model will not adapt very well to the training data. On the other hand we risk over-fitting by using a too high order and the models will be less able to generalise.

Smoothing The language model implementation in SRILM also features several smoothing algorithms. Smoothing is done in order to minimise the problem of estimating the probability for n-grams that were not in the training data.

We conducted several test where we varied adjusted all of these parameters in order to find the configuration most suited for our use.

4.5.1 Collecting Test and Training Data

Our approach requires two language models, and before we can generate those we must acquire some data to train them on. This is done by selecting article sections from our dump by comparing them to a clean and dirty heuristic. Before defining those we examined all articles in order to find the distribution of section headings and their average length. The motivation for doing so was the expectation that there is a correlation between the length of a section and whether or not it is clean. Heading levels were taken into account in this survey, so the headings created by the wiki markup `== Notes ==` and `=== Notes ===` (second and third level heading "Notes") were counted separately. This list turned out to be quite handy and it was referenced directly by the script we developed for collecting the training data.

The similar structure of many of the articles means that often-repeated section headings are good indications of sections being noise (e.g. "References" and "Notes"). A intuition that also Corpus Clean based its classification on. Intuitively, the converse also seemed true: Rare headings are usually clean. In addition we also expected clean sections to be longer, especially after the removal of noise-introducing templates (see Section

3.3) and the preproccesing (described in Section 4.4.1 above) as both will normally discard more content from dirty sections than from clean ones.

When developing our heuristics we had to strike a balance between quantity and quality. Using a greedy heuristic would get us a large amount of training data, but it would contain more impurities than the sets collected by a less greedy heuristic. After we had done some preliminary experimentation we did some additional tuning to find this balance.

The format of the test and training data is the same as the one used during classification (described in Sections 4.4.1 and 4.4.2), except that we do not add the sentence-start and sentence-end tokens as these are added implicitly by SRILM during training.

4.5.1.1 The Heuristics

We will start by detailing the original heuristics. The sections included in the “clean set” matched the following criteria:

- It did not include any lists
- and it had more than four paragraphs
- and it had a unique heading

Training material for the dirty model consisted of article sections with the following qualities:

- It had one of the following-second level headings (frequencies in parentheses):
  - Works (13823)
  - Deaths (11372)
  - Births (10832)
  - Publications (8232)
- Or:
  - it had a heading that is used more than 5000 times
  - and the average length of sections with this heading is less than 750 characters.

The reason for creating special rules for some headings was that they were in the top 40 most frequently used headings, but they were on average a bit too long (around 1000 characters) to be caught by the heuristic rules.

Manually inspecting the data collected from the original heuristics revealed that there some clean sections had made it into the dirty set. To mitigate this we created a new set of heuristics that we called "the refined heuristics". The difference between the refined heuristics and the first set of rules (which we now refer to as "the original heuristics") is that we added a blacklist of all the headings we found in the dirty set that were not obviously dirty. In addition we no longer included sections with the heading "Works". The headings added to the blacklist were:

- Awards
- Geography
- Education
- Transportation
- Personal life
- Family
- Trivia
- Results
- People
- Achievements
- Transport

4.5.1.2 The Data Sets

Table 4.2 shows the different data sets with their "clean" (C) and "dirty" (D) sub-sets that were used in the training and evaluation of the classifier. The sets train_A, train_B and train_C were extracted from the dump using the rules discussed above, and the smaller training sets are proper subsets of the larger ones. The sets test_A and test_B were by setting aside some of the material extracted in the creation of the training sets and were used to measure the predictive powers of the models as well as the performance of the classifier.

When querying the CDB (the storage facility employed by mwlib) for articles it returns a list of names that has several long spans that are sorted alphabetically. Since there are several naming conventions in use on Wikipedia, as for instance the articles named "List of" that mostly share

Table 4.2: Training and test sets

Class	Set	File size	Sections	Description
C D	train_A	1,100 MB	208,000 1,706,000	the largest training set
C D	train_B	700 MB	139,000 1,124,000	about two thirds of train_A
C D	train_C	350 MB	69,000 562,000	about a third of train_A
C D	test_A	120 MB	23,000 190,000	for evaluating the models
C D	test_B	15 MB 2 MB	3,000	for evaluating the classifier
C D	the gold standard	500 KB 100 KB	243 63	annotated by tree annotators
C D	the silver standard	800 KB 200 KB	273 144	annotated by a single annotator

a similar structure, it is important to stratify the training data. This is to ensure that none of the sets created from the data have different compositions. Our script for extracting the data sets randomised the list of articles before it started collecting sections.

As seen in Table 4.2 number of sections in the "dirty" sub-sets greatly outnumbers the sections in the corresponding "clean" sub-set in the sets that are balanced on size (i.e train_A, train_B, train_C and test_A). This is a result of sections that consists of relevant linguistic content usually being longer than sections that mostly consists of noise. The "template processing" stage (detailed in Section 3.3) also removes far more content from the noisy sections, and the same applies for the preprocessing for the classification (detailed in Section 4.4.1).

The same can be observed by examining file sizes for the "clean" and "dirty" components for test_B. This set was compiled so that it would have the same number of sections of each class. This was in order to test the classifier without having to do any weighing when analysing the results.

The gold standard were created by manually classifying sections as clean or dirty. The gold standard was created by collecting sections from test_B that were mistakenly classified at least once by two differently

configured classifiers (detailed in Section 4.5.2). As a result this set contains sections that are particularly difficult as can be seen by the results from our experiments below. The sections in this set were manually classified by three human annotators any disagreements were resolved by using the classification reached by the majority. During the annotation it became apparent that not only were some sections difficult to classify even for humans, but that our notion of relevant linguistic content was not developed fully. We will discuss some examples of difficult sections in Section 4.6 below.

The sets described so far only contain sections that were collected by our heuristic rules. While experimenting on those did give us some indication on how good or bad the different configurations are, their composition are not like most articles in the dump. In order to find how the classifier would perform in practice one annotator classified all sections from a hundred randomly chosen articles creating what we call the silver standard.

4.5.2 Experiments

Having decided to take a similar approach to content classification as NCLEANER (Evert, 2008) and to base our approach on SRILM (Stolcke et al., 2011) we now needed to find the best possible parameters for our n-gram models. We also needed to know if the best configuration performed well enough for our purpose. Even though NCLEANER got decent results in the CleanEval shared task it was not guarantied that the performance would be on the same level with our implementation on different domains and data format.

Our experiments tested the following configuration parameters: Type and quantity of training data, n-gram order and smoothing algorithm. These tests were carried out on several test-sets namely test_A, test_B, the gold standard and the silver standard (all described in Section 4.5.1.2). These data sets all have different properties.

4.5.2.1 Evaluation Metrics

Before assessing the results of our experiments we need to establish suitable metrics as well as methods of comparing them. We will use perplexity (introduced in Section 4.2.1), F-measure and the sign test when reporting and discussing our results (both detailed in the immediately following sections).

F-Measure We use F-measure (or F_β-score) to give us an idea of how well a given configuration is at identifying clean and dirty sections. The F_β-score is a measure of how well a system is at identifying items from a target class. Since we are classifying sections the target class can either be clean sections (if we view our system as a "clean classifier") or dirty sections (if we view our system as a "dirty classifier"). Before describing how we calculate the F-measure we need to define the following terms:

True Positives (TP): The number of items correctly identified as the target class.

False Positives (FP): The number of items wrongly identified as the target class.

False Negatives (FN): The number of items in the target class that were not identified.

We use these values to find the precision (P) and recall (R), which is defined as:

$$P = \frac{TP}{TP + FP} \quad (4.8) \qquad R = \frac{TP}{TP + FN} \quad (4.9)$$

Using different values for β in the F_β-score results in a different weighing of weighing precision and recall. We use a value of 1 which puts equal importance on both. When $\beta = 1$ the F-measure is given by the harmonic mean of precision and recall[4]:

$$F_1 = \frac{2PR}{P + R} \quad (4.10)$$

(Jurafsky and Martin, 2009, p. 489).

Our classifier can both be viewed as a system that identifies clean sections and as system that identifies dirty sections, and which configuration gets the highest score can depend on which view we take when calculating the F-measure. When reporting the results of our experiments below we calculate the F_β-score both ways and report the macro average.

Sign Test When finding the optimal combination of training set, order and smoothing algorithm it is often useful to tell if one configuration is significantly better than the other. We use the sign test to determine significance.

[4]For other values of β: $F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R}$.

When comparing configuration A and configuration B we do a pairwise comparison of the outcomes (a list like $(a_1, b_1), (a_2, b_2), \ldots$). The null hypothesis is that there is no difference between them. In that case, A and B both have the same probability of having made the right classification for any pair that differs. The alternate hypothesis is that configuration A is better and that for each pair with different classifications the probability for configuration A being correct is greater than the probability of B being correct. The null hypothesises (H_0) and the alternate hypothesis (H_0) can be expressed as:

$$H_0 : P(Correct\ A) = P(Correct\ B) = 0.5 \tag{4.11}$$

$$H_1 : P(Correct\ A) > P(Correct\ B) \tag{4.12}$$

If the null hypothesis is correct, the outcomes will have a binomial distribution. The significance probability (p) for configuration A having w correct outcomes out of m with a base probability k is found by summing up the probability of getting w or more correct outcomes:

$$p = \sum_{i=w}^{m} \binom{m}{i} k^i (1-k)^{m-i} \tag{4.13}$$

(Johnson and Bhattacharyya, 2010, p. 591-596). We use the significance probability of 0.05 to determine if the difference between to configurations is significant.

The sign test is applicable to any set of observations as long as they can be compared pairwise. However, since pairs with similar outcomes are discarded the effective sample space is reduced. Our classifier returns a positive or negative real number for wanted and unwanted content which could be viewed as a measure of how confident the classifier is in its decision. Unfortunately, high confidence does not mean that a given classification is actually better. We can therefore not use this in order to employ tests that are statistically more powerful like for instance the Wilcoxon signed-rank test.

Listing 4.3 on the next page shows a Python script that uses the sign test to check for significance. The tuple `outcomes` contains the pairwise classifications made by configuration A and B, these are boolean values that are true for correct classifications and false otherwise. Only the outcomes that differs for a and b are included in the test (line 7). The variables `m` and `w` will at the end of the loop hold the numbers of different outcomes and the number of correct outcomes for configuration A. In line 12 we use Equation 4.13 to find the likelihood of doing equally or better with a coin-toss by summing up the probability of getting `w`-out-of-`m` or more correct outcomes with a probability 0.5.

Listing 4.3: The sign test implemented in Python

```
1  import scipy.stats
2
3  k = 0.5
4  outcomes = ((a1, b1), (a2, b2), ...)
5
6  for a, b in outcomes:
7      if a != b:
8          m += 1
9          if a:
10             w += 1
11
12 p = scipy.stats.binom.sf(w - 1, m, k)
```

4.5.2.2 Experimental Setup

We tested classifier performance on the test sets detailed in Section 4.5.1.2. They were build for the combinations of different training sets (also detailed in Section 4.5.1.2), different n-gram orders (2, 3, 4, 5 and 10) and the smoothing algorithms detailed below. When describing these algorithms we will use the form used by Chen and Goodman (1998) with some modification so they match the notation used elsewhere in this thesis.

Add one

$$P_{+1}(w_i|w_{i-n+1}^{i-1}) = \frac{1 + C(w_{i-n+1}^{i})}{|V| + C(w_{i-n+1}^{i-1})} \tag{4.14}$$

As shown in Equation 4.14 add one smoothing modifies the probability of an n-gram by adding a constant of one to its count. In order to keep the sum of probabilities from exceeding one the size of the vocabulary (V), that is number of different events observed, is added to the count of the $n-1$ events in the denominator. This is an uncomplicated way of ensuring that all estimates, including those of n-grams not seen in the training data, will be greater than zero. A downside of this approach can be that the too much of the probability mass is moved to unseen n-grams.

Witten-Bell (Witten and Bell, 1991).

$$\begin{aligned} P_{WB}(w_i|w_{i-n+1}^{i-1}) = \lambda w_{i-n+1}^{i-1} P(w_i|w_{i-n+1}^{i-1}) + \\ (1 - \lambda w_{i-n+1}^{i-1}) P_{WB}(w_i|w_{i-n+2}^{i-1}) \end{aligned} \tag{4.15}$$

Where P is the maximum likelihood estimation as presented in Equation 4.6 and $1 - \lambda w_{i-n+1}^{i-1}$ is defined as:

$$1 - \lambda w_{i-n+1}^{i-1} = \frac{N(w_{i-n+1}^{i-1}\bullet)}{N(w_{i-n+1}^{i-1}\bullet) + C(w_{i-n+1}^{i})} \tag{4.16}$$

In Equation 4.16, N is the number of unique n-grams matching a given sequence, where the symbol $\bullet$ matches any event. The probabilities from $P(w_i|w_{i-n+1}^{i-1})$ and $P_{WB}(w_i|w_{i-n+2}^{i-1})$ are weighed by Equation 4.16, which is close to one for low frequency n-grams and close to zero for high frequency n-grams. For unseen n-grams, only the estimation from the lower order model is used.

This smoothing algorithm was developed for statistical data compression. This is a task where the tokens operated on are usually fixed-length byte sequences instead of words. This might be an advantage for character level models.

Constant Discount (Ney and Essen, 1991).

$$P_{CD}(w_i|w_{i-n+1}^{i-1}) = \frac{max(C(w_{i-n+1}^{i}) - D, 0)}{C(w_{i-n+1}^{i-1})} + (1 - \lambda w_{i-n+1}^{i-1})P_{CD}(w_i|w_{i-n+2}^{i-1}) \tag{4.17}$$

Where D is the discount and the interpolation factor $(1 - \lambda w_{i-n+1}^{i-1})$ is:

$$1 - \lambda w_{i-n+1}^{i-1} = \frac{D}{C(w_{i-n+1}^{i-1})} N(w_{i-n+1}^{i-1}\bullet) \tag{4.18}$$

The discounting parameter (D) is subtracted from all counts and the interpolation factor in Equation 4.18 is an estimate for the probability of the n-gram being unknown. We use a discounting constant of one in our experiments.

The models smoothed with constant discount were a lot smaller than similar combinations of the same n-gram order and amount training data. This made it possible to run tests using 15-gram models for this smoothing method. The reduction in size was a result of setting the discounting parameter to one, which led to all n-grams that were only seen once during training to have their count discounted to zero (Ney and Essen, 1991, p. 826), which in turn cased SRILM to discard them.

Kneser-Ney (Kneser and Ney, 1995).

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{max(C(w_{i-n+1}^{i}) - D, 0)}{C(w_{i-n+1}^{i-1})} + (1 - \lambda w_{i-n+1}^{i-1}) \frac{N(\bullet w_{i-n+2}^{i})}{N(\bullet w_{i-n+2}^{i-1} \bullet)} \tag{4.19}$$

Where the interpolation factor is the same as for constant discount (Equation 4.18). Kneser-Ney smoothing is similar to constant discount except for that the probability is interpolated with a value determined by how many different events have been observed after w_{i-n+2}^{i-1} SRILM does have a command line parameter for the discounting value for Kneser-Ney smoothing. It is possible to estimate this during training by letting $D = \frac{n_1}{n_1+2n_2}$ where n_1 and n_2 are the number of events that occurred once and twice during training (Ney et al., 1994; Chen and Goodman, 1998).

Modified Kneser-Ney (Chen and Goodman, 1998). This is an modification of Kneser-Ney where the discount value varies for depending on how often a given n-gram is seen. The interpolation factor is changed to:

$$1 - \lambda w_{i-n+1}^{i-1} = \frac{D_1 N_1(w_{i-n+1}^{i-1} \bullet) + D_2 N_2(w_{i-n+1}^{i-1} \bullet) + D_{3+} N_{3+}(w_{i-n+1}^{i-1} \bullet)}{C(w_{i-n+1}^{i-1})} \tag{4.20}$$

Where D_1, D_2, and D_{3+} are the discount for n-grams seen once, twice or more and N_1, N_2, and N_{3+} are the number of unique seen one, twice or more.

4.5.2.3 Results

Figure 4.3 shows the perplexity from different configurations of our classifier on test_A that is a set created with the same heuristics as the training data and balanced so the “clean” and “dirty” parts are roughly the same size. Perplexity is measured separately on the “clean” and “dirty” sections of the test set and the results was then averaged. The most striking observation in this plot are the high perplexity given by the higher order add one-smoothed models. We should note that the extent of this increase is visually exaggerated by the step in n-gram order from 5 to 10. The other smoothing methods are almost indistinguishable. It is also interesting to see that the perplexity remains stable for the different training sets. This is probably because even the smallest of our training sets, train_B, is quite large with a size of 350 MB.

Figure 4.3: Perplexity on test_A

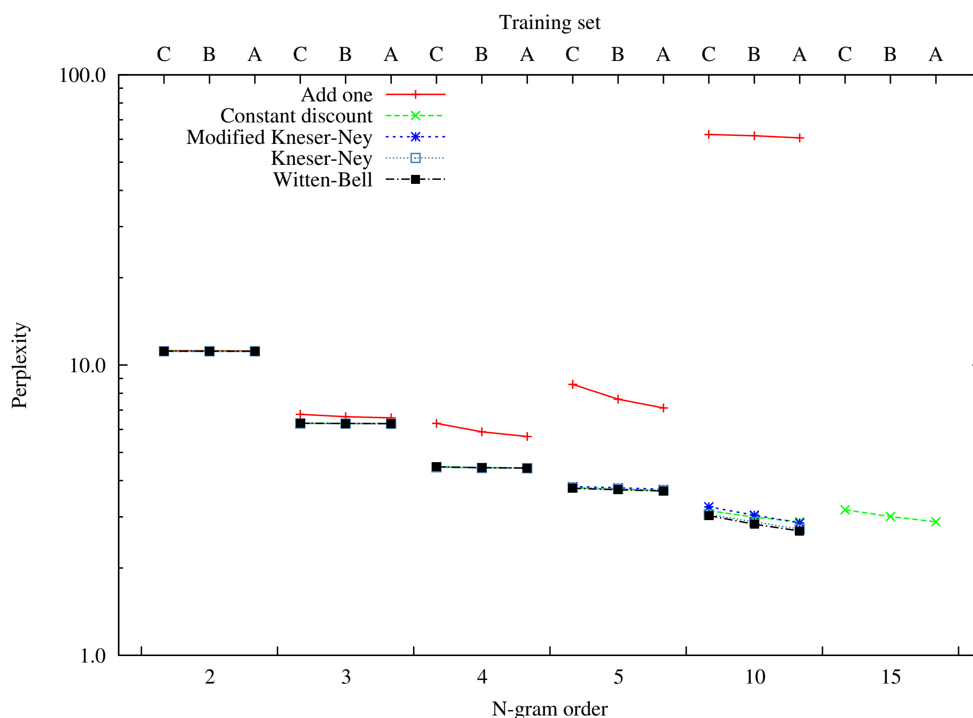


Figure 4.4: F1-score on test_B

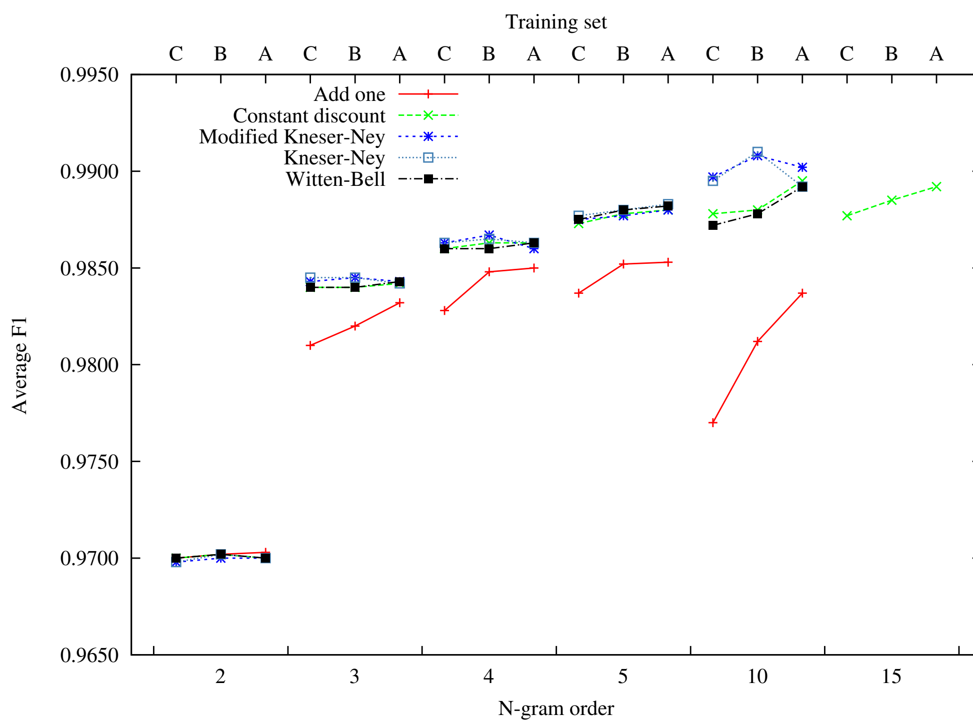
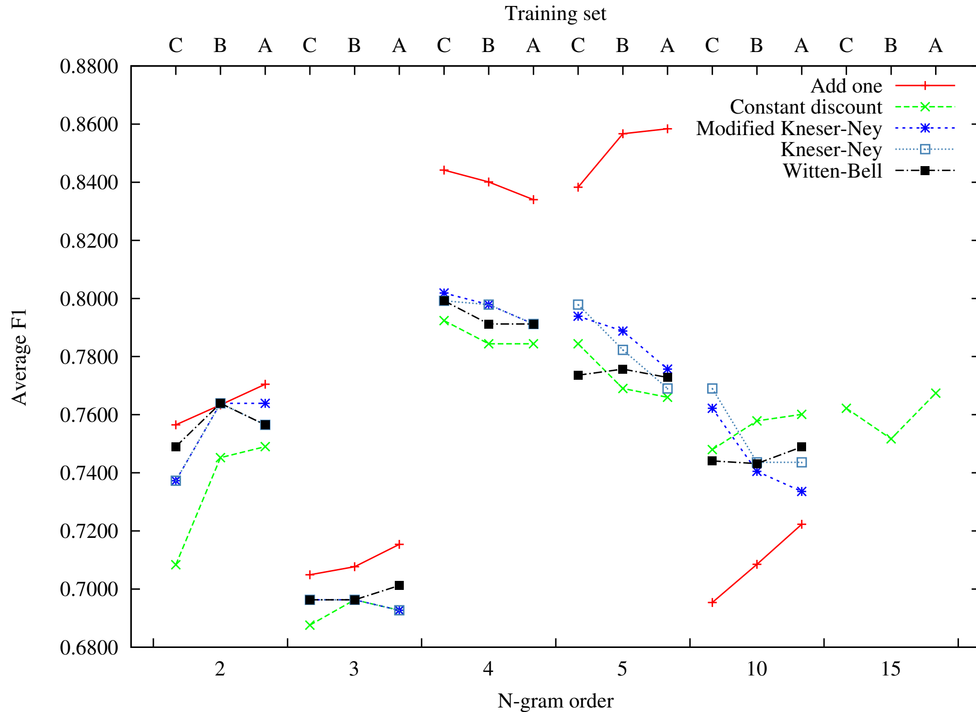


Figure 4.5: F_1-score on the gold standard

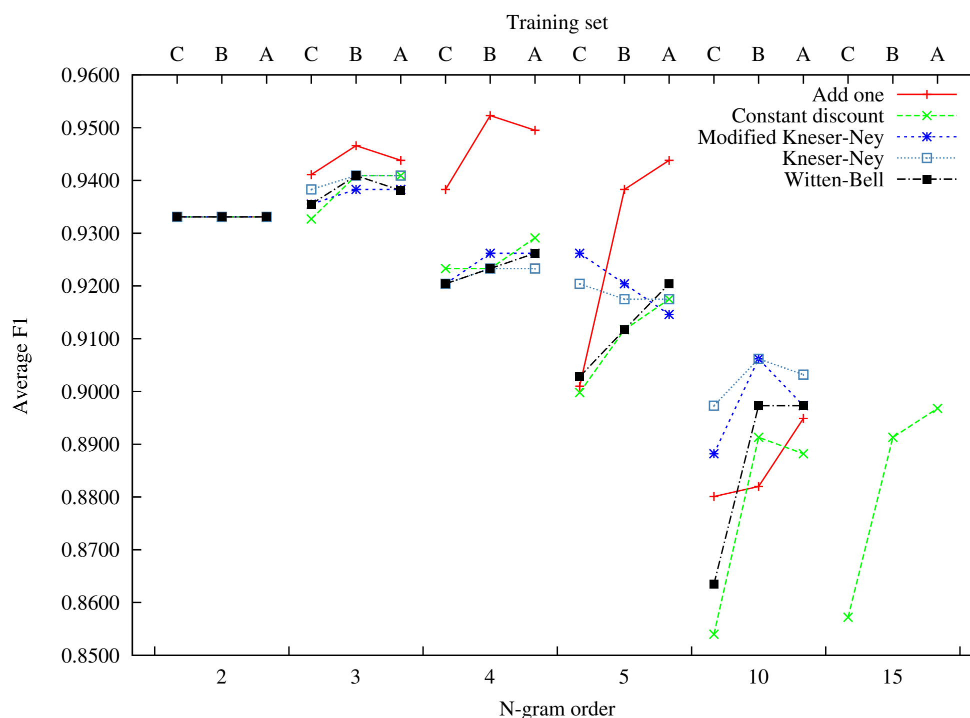


The macro averaged F_1-scores for different configurations on the set test_B is shown in Figure 4.4. Test_B was created in the same way as the training sets, but is balanced so it has 3,000 sections of each class. As in Figure 4.3, add one stands out with noticeably lower scores for configurations with an n-gram order three or higher. Except for the configurations using add one, we once again see that the amount of training data have little effect on the lower order models.

The scores on this test set are extremely high, but unfortunately they do not reflect the actual performance of the classifier. The sections in test_B are very similar to those in the training sets since they were all created using the same heuristic method and do not reflect the actual composition of Wikipedia.

The gold standard is one of the two test sets have been hand corrected by human annotators. This set consists of sections that that were collected using the original heuristics (the revision of the heuristics is detailed in Section 4.5.1.1) and that wrongly classified by at least one configuration. The sections in this set are considerably harder than the sections in test_B and the silver standard and this can be seen by the overall lower score shown

Figure 4.6: F_1-score on the silver standard



in Figure 4.5.

Add one generally out-performs the other smoothing methods on this set except for the configurations using models with an n-gram order 10. Classifiers using trigram models seem to do worse than those using models of n-gram order two and tree. This is likely to be a result of the composition of this set. The gold standard was created from sections that one of two candidate classifiers using 15-gram and trigram models got wrong. This plot seems to suggest using add one smoothing and an n-gram order of four or five. We will return to this test and do significance testing on the results in Section 4.5.2.4.

Figure 4.6 shows the performance on the silver set, which was created by manually annotating the sections for a hundred randomly chosen articles. The silver standard is the set that is likely to most similar to the rest of the dump as it contain was not extracted using the heuristic rules.

Yet again the highest scoring configuration uses add one smoothing. For most of the smoothing algorithms the performance drops steadily when the n-gram order increases beyond three. The configurations using bigram models all get the same F_1-score on this test, but their classifications differ

on individual sections.

4.5.2.4 Discussion

We have run tests with multiple classifiers on different sets in order to pick the one with the best performance for our system. Our test sets have very different properties and as one might expect a different configuration comes out on top for each of them. We will briefly summarise our finding below and we argue for our choice of parameters for our classifier.

When testing on test_A and test_B (shown in Figures 4.3 and 4.4) there are relatively small improvements from increasing the amount of training data. On the gold and silver standards there is no apparent correlation between the size of the training set and performance. In fact, several configurations trained on the smallest training set (train_C) outperform similar configurations trained on larger sets (i.e. train_B and train_A). We should keep in mind that these sets are fairly small and that a change of just a few different classifications is clearly visible on these plots. Take for instance the dropping performance of 5-gram models smoothed with modified Kneser-Ney (shown in blue) in Figure 4.6 as the amount of training data increases. This is a decrease in average F_1-score is from 0.9262 to 0.9146, that is the result of the classifier making four extra mistakes.

We put more emphasis on the results on the silver and gold standards then on the sets generated from the heuristic rules as their distribution of relevant linguistic content and noise is hand made. In addition the silver standard should match the rest of Wikipedia as it was collected from randomly selected articles. On all of our experiments, the results from the classifiers using add one smoothing are the most eye catching. They perform poorly on test_A (shown in Figure 4.3) and test_B (shown in Figure 4.4), but they often get the best scores on the manually annotated sets (shown in Figures 4.5 and 4.6).

Table 4.3 and Table 4.4 displays the best performing configurations organised by n-gram order for on the gold and silver standard. They also include the best configuration that does not use add one smoothing for the n-gram order of the best performer for comparison. The rightmost column shows the significance probability resulting from comparing it with the best configuration using the sign test (detailed in Section 4.5.2.1). As on the plots, the performance of add one smoothed models stands out.

We use add one smoothed 4-gram models that are trained on the largest training set (train_A) in our system. This decision is motivated by its good performance on both the gold and silver standard. It is outperformed by similar configurations using less training data on both sets,

Table 4.3: Significance for the best performing (F_1: 0.8584) configuration (add one smoothed 5-gram models trained on train_A) on the gold standard.

Order	Training set	Smoothing	F_1-score	Significance
2	train_A	Add one	0.7705	<0.05
3	train_A	Add one	0.7154	<0.05
4	train_C	Add one	0.8442	-
10	train_C	Kneser-Ney	0.7690	<0.05
15	train_A	Constant discount	0.7674	<0.05

Table 4.4: Significance for the best performing (F_1: 0.9523) configuration (add one smoothed 4-gram models trained on train_B) on the silver standard.

Order	Training set	Smoothing	F_1-score	Significance
2	any	any	0.9331	-
3	train_B	Add one	0.9466	-
5	train_A	Add one	0.9438	-
10	train_B	Kneser-Ney	0.9062	<0.05
10	train_B	Modified Kneser-Ney	0.9062	<0.05
15	train_A	Constant discount	0.9868	<0.05

but the differences in performance is minimal. Our chosen configuration is outperformed on the gold standard by 5-gram models using add one smoothing. Since we feel that the silver standard represents the content of our snapshot better we put more weight on the results on this set.

Admittedly we did not expect add one smoothing to perform this well compared to the other, more sophisticated, smoothing techniques we pitted it against. Besides for the sense of completeness, our motivation for including add one in our tests was this brief mention of it by Stolcke et al. (2011):

> The latter [additive smoothing] is useful for instructive purposes, and can give better results when the goal is not to minimize model perplexity (e.g., when using N-gram models for classification tasks).

Add one smoothing was clearly outperformed by several of the smoothing algorithms we employed (absolute discount, Kneser-Ney and modified

Table 4.5: Classifier performance with the original and refined heuristics

	Original		Refined	
Order	Test_B^o	Gold standard	Test_B^r	Gold standard
2	$0.9753^{B+\ C+}$	$\mathbf{0.7426}^{A+}$	0.9703^{B+}	0.7705^{A+}
3	$0.9797^{Bw\ Cw}$	0.6002^{B+}	$0.9845^{Bk\ Bkk\ Ck}$	0.7154^{A+}
4	0.9828^{B+}	0.6948^{D+}	0.9867^{Bkk}	0.8442^{C+}
5	0.9848^{Bw}	0.7109^{C+}	0.9883^{Ak}	$\mathbf{0.8584}^{A+}$
10	0.9878^{Bk}	0.6208^{B+}	$\mathbf{0.9910}^{Bk}$	0.7690^{Ck}
15	$\mathbf{0.9880}^{Bc}$	0.5774^{Ac}	0.9892^{Ac}	0.7674^{Ac}

$^+$ Add one ^{c} Constant discount ^{w} Witten-Bell ^{k} Kneser-Ney
^{kk} Modified Kneser-Ney ^{A} Train_A ^{B} Train_B ^{C} Train_C ^{D} Train_D

Kneser-Ney) in the experiments performed by Vatanen et al. (2010), where they used character level n-gram models for language identification. Although, both the size of training data (the U.N. Universal Declaration of Human Rights) and the size of the text segments that they classified (5-21 characters) were significantly smaller than what we used for our experiments.

4.5.2.5 Evaluating the Heuristics

As previously mentioned in Section 4.5.1.1 we fine-tuned the heuristics used for collecting test and training data in order to make these sets "purer" (i.e. representative of the distribution we want to model). Purer sets came at the price of smaller size and the largest training set (train_A) shrank from 1.5 GB to 1.1GB.

The best performing configurations for both the original and revised heuristics are shown in Table 4.5. The scores shown are the macro average of F_1 for "clean" and "dirty" sections. Test_B^o and test_B^r are created with the original and revised heuristics respectively. The set train_D was a small training set used during our initial rounds of experimentation. It had a size of 150MB and consisted of sections extracted with the original heuristics.

Interestingly, the n-gram order for the best performing classifiers trained on the original heuristics for gold standard and test_B are at the opposite extremes. This could be a sign of the models learning how to predict the "impure" training data, which is a drawback when operation on the gold standard.

There is an improvement in the performance on test_B with the revised

heuristics. There are two possible factors behind this: There are no longer any sections included in the wrong part of test_B that the classifier gets penalised for getting right. That is, the training sets and test sets are now more uniform and easier for the classifier. The other factor is an actual improvement in performance. If we compare the scores on the gold standard, where an increased score does mean an actual performance gain, we see that trading quantity for quality paid off. We only used the refined heuristics from that time on and all the results reported from our classification experiments use the revised heuristics unless otherwise stated.

4.6 Revisiting Relevant Linguistic Content

As mentioned in Section 4.5.1.2 we created a set with gold standard section classification. The content of this set was selected by two classifier with an n-gram order of 3 and 15 on test_B^o, which was created from the original heuristics and collecting the sections that were wrongly classified at least once.[5] We also added 10% extra sections to serve as controls.

Listing 4.4: A clean section

```
<h1>Catharose de Petri</h1>
<p><Strong>Catharose de Petri</Strong> (real name Henriette
   Stok Huyser 1902-1990) was a Dutch born mystic and
   co-founder of the <ArticleLink>Lectorium
   Rosicrucianum</ArticleLink>, an international esoteric
   school based on <ArticleLink>Gnostic</ArticleLink> ideas
   of <ArticleLink>Christianity</ArticleLink>.</p>
<p>Catharose de Petri founded the Lectorium in
   <ArticleLink>1935</ArticleLink> with two other Dutch
   mystics, <ArticleLink>Jan van Rijckenborgh</ArticleLink>
   and his brother Zwier Willem Leene after meeting them as
   a member of the Dutch branch of <ArticleLink>Max
   Heindel</ArticleLink>'s <ArticleLink>Rosicrucian
   Fellowship</ArticleLink>. The three broke away from
   Heindel's interpretation of the
   <ArticleLink>Rosicrucian</ArticleLink> message to form
   their own movement, the <Emphasized>Lectorium
   Rosicrucianum</Emphasized>.</p>
[... snip]
```

Three human annotators classified each section as clean or dirty. Any conflicting classifications were resolved by voting. The annotators were not aware of which part of test_B the sections originally came from. The

[5] Our first strategy for choosing n-gram parameters involved studying two different configurations. This was then abandoned for the approach detailed in Section 4.5.

Listing 4.5: A section containing both clean and dirty text

```
1  <h2>Cutting-Edge Research</h2>
2  <p>The hospital's activities include comprehensive research
     in the laboratory, in the treatment setting, and in the
     community throughout the hospital and its research
     institutes.</p>
3  <p>The majority of the basic laboratory research conducted
     at the hospital takes place in <NamedURL>The Saban
     Research Institute</NamedURL>.</p>
4  <p>The Saban Research Institute is the largest and most
     productive pediatric research center in the western
     United States, ranking fifth in the nation in federal
     funding for pediatric research at stand-alone pediatric
     facilities, including from federal agencies, like the
     <ArticleLink>National Institutes of Health</ArticleLink>
     and the <ArticleLink>Centers for Disease Control and
     Prevention</ArticleLink>.</p>
5  <p><Strong>Current Research Programs Include:</Strong></p>
6  <p>• <NamedURL>Cancer Research</NamedURL></p>
7  <p>• <NamedURL>Cardiovascular Research</NamedURL></p>
8  <p>• <NamedURL>Community Health Outcomes and Intervention
     Research</NamedURL></p>
9  <p>• <NamedURL>Developmental Biology</NamedURL></p>
10 <p>• <NamedURL>Gene, Immune, & Stem Cell
     Therapy</NamedURL></p>
11 <p>• <NamedURL>The Stem Cell Project</NamedURL></p>
12 <p>• <NamedURL>Imaging Research</NamedURL></p>
13 <p>• <NamedURL>Microbial Pathogens</NamedURL></p>
14 <p>• <NamedURL>Neuroscience</NamedURL></p>
15 <p>• <NamedURL>Institute for the Devloping
     Mind</NamedURL></p>
```

annotation was split into two runs. In the first run, the calibration run, only 10 % of the set was classified. The purpose of this was to discover any different views on what is relevant linguistic content by the annotators. The inter-annotator agreement for the set as a whole was 88.9%.

Recall from Section 1.1.1 that the content we wish to retrieve are spans of text that contain information about the subject matter of the article and that have a form that requires grammatical analysis for interpretation. Listing 4.4 shows the first half of Section 1 in the calibration set, which easily fulfils the requirements for being relevant linguistic content.

The section shown in Listing 4.5 starts out as clean, but half of it (starting from line 5) is simply a list of links. A result of using sections as the smallest unit that is classified is that we must occasionally chose

between discarding relevant linguistic content or keeping some noise. In these cases it comes down to how much dirt we are willing to put up with in order to get the clean content. There are also some peculiar use of markup in this section like using paragraphs and the bullet character to create a list, but this is to be expected due to the diverse group of non-experts that maintains the articles on Wikipedia. This section was deemed clean by the annotators.

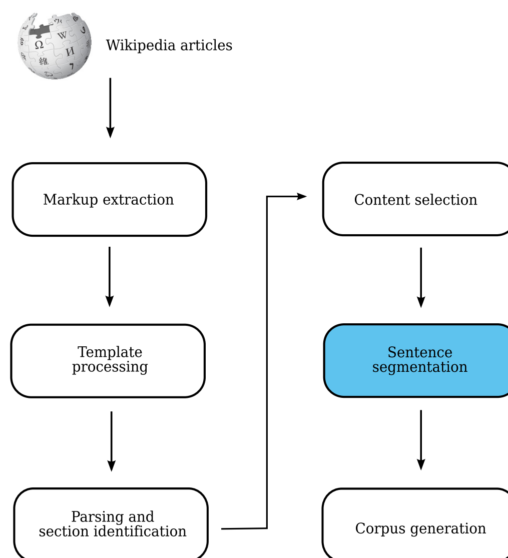
Chapter 5

Sentence Segmentation

This chapter details the "Sentence segmentation" stage in our system. As can be seen in Figure 5.1 this is the last stage before the actual corpus generation. At this point, articles are already parsed and split into sections, and the sections with little relevant linguistic content are marked as such.

Throughout this thesis we use the term "sentence" when we refer to a unit of text that is convenient for further language processing. While this will often be a sentence in the strict grammatical meaning of the word, it also includes things like a single-word headings, list items and so on. The definition offered for the term "text-sentence" by Nunberg (1990, p. 22) seem to be good description of the units of text we want to identify:

Figure 5.1: Overview of our system



> But none of them [other definitions of "sentence"] deals with what we will call the "text-sentence," which is the fundamental unit of text-grammatical structure (and not incidentally, of instruction of writing. The text-sentence is that unit of written text that is customary presented as bracketed by a capital letter and a period (though those properties are not criterial).

The reason for performing sentence boundary detection during the creation of the corpus is that most NLP tools expect their input to be separated into sentences. Additionally, this is one of the many types of language processing that benefits from the presence of markup and, more generally, access to layout information so it makes sense to do this while we still have access to all of the markup elements.

The generation of the GML markup that makes it into the corpus is done immediately after sentence boundary detection. Since we use a combination of heuristic rules and a third-party tool that is not markup-aware, the sentence boundaries returned from this tool must be merged with the internal representation used in our system. We will introduce an algorithm for doing this in Section 5.3.

5.1 Choosing a Sentence Segmenter

In order to find a sentence boundary detector suitable for our setup, we tested several candidate systems. They are shown in Table 5.1 where they are labelled with their general approach to finding sentence boundaries: rule based (R), unsupervised (U) or supervised (S). With the exception of GATE (that can operate on HTML), these tools are not markup aware. The sentence splitter in LingPipe has two configurations, one tuned for "general text" and one tuned for the bio-medical domain. We test them both and refer to them as "LingPipe (general)" and "LingPipe (medical)". We used the implementation and the pre-trained model in NLTK (Bird et al., 2009) for Punkt.

With the exception of the GATE sentence splitter, all of these tools have already been tested on several corpora including the WeScience corpus (Ytrestøl et al., 2009) in an earlier collaborative effort (Read et al., 2012). Table 5.2 shows the character level F_1-score for the best performing configuration for each system on different corpora. Besides WeScience (WS), the corpora used in these experiments were: The Brown Corpus (Francis and Kucera, 1982), the Conan Doyle Corpus (CDC) (Morante and Blanco, 2012), the GENIA Corpus (Kim et al., 2003), Wall Street Journal

Table 5.1: Candidate systems

System	
CoreNLP^R	`http://nlp.stanford.edu/software/corenlp.shtml`
GATE^R	`http://gate.ac.uk`
LingPipe^R	`http://alias-i.com/lingpipe/`
MxTerminator^S	Reynar and Ratnaparkhi (1997), `ftp://ftp.cis.upenn.edu/pub/adwait/jmx/`
OpenNLP^S	`http://opennlp.apache.org`
Punkt^U	Kiss and Strunk (2006), `http://nltk.org/api/nltk.tokenize.html`
Splitta^S	Gillick (2009), `http://code.google.com/p/splitta`
RASP^R	Briscoe et al. (2006), `http://ilexir.co.uk/applications/rasp/`
Tokenizer^R	`http://www.cis.uni-muenchen.de/~wastl/misc/`

^{R} Rule based
^{S} Supervised machine learning
^{U} Unsupervised machine learning

Table 5.2: Results from our earlier experiments (Read et al., 2012). Only the best result from the different setups for each system is displayed.

	Brown	CDC	GENIA	WSJ	WS	WNB	WLB
CoreNLP	93.6	98.3	99.0	94.8	97.9	96.4	90.9
LingPipe (general)	96.6	99.1	98.6	98.7	98.1	96.1	94.2
LingPipe (medical)	94.5	97.2	**99.8**	90.9	98.0	95.6	94.5
MxTerminator	96.5	98.6	98.5	98.5	97.2	95.9	92.2
OpenNLP	96.6	98.6	98.8	99.1	97.9	96.5	92.0
Punkt	96.4	98.7	99.3	98.3	97.7	96.7	94.5
RASP	96.8	99.1	98.9	99.0	99.1	96.6	94.6
Splitta	95.4	96.7	99.0	**99.2**	98.9	95.5	93.4
Tokenizer	**96.9**	**99.2**	98.9	**99.2**	**99.2**	**96.8**	**94.9**

(Marcus et al., 1993) (WSJ) and web blogs in the NLP (WNB) and Linux (WLB) domains from the WeSearch Data Collection (Read et al., 2012).

One of the tools, Tokenizer, outperformed the others on all corpora except for the GENIA Corpus. This corpus is in the biomedical domain for which the best performing system, LingPipe (medical), is tuned for. For those corpora that contained markup (WeScience, Linux blogs and NLP blogs) all tools benefited from forcing sentence breaks around selected markup directives. This improvement was most distinct on WeScience where the average increase in F_1-score was almost nine points (as opposed to 1.2 and 1.8 points on average for WNB and WLB). The relatively high density of markup directives in Wikipedia articles is a likely cause for this, as some sentence boundaries are not terminated by punctuation, but are separated from the rest of the text by its appearance or placement (e.g. headings and list items). These are almost impossible to detect by only examining the text itself.

The reason for running new tests on these tools is that our objective is to find the best fit for our specific setup and not to give general measures of performance on marked up text. The practical differences between our current setup and that used earlier will be detailed below in Section 5.1.1.

5.1.1 Experimental Setup

We removed all markup from the text before running it trough the boundary detectors. Any attempts at benefiting from the presence of markup was done as pre and post -processing steps. The WeScience corpus (introduced in Section 2.2.2) has gold standard sentence boundaries and was created from the same Wikipedia snapshot that we are using in this work. This makes it an invaluable resource for evaluating sentence segmenters. We ran all the candidates shown in Table 5.1 on pure-text versions of WeScience created from the original article markup. The sentence boundaries were then compared with the gold standard in WeScience.

Although we did not consider using Corpus Clean (Ytrestøl et al., 2009) for detecting sentence boundaries, we did include it in our test as we felt it would be informative to compare its performance to that of the other systems. Our approach for measuring its performance was slightly different than for the other tools. We created segmented pure-text by running it on the same set of articles that are in WeScience and performing the same conversion to pure text as we did for the gold standard sentence boundaries (detailed below in Section 5.1.1.1). It should be noted that since our gold standard was created by manually correcting the sentence breaks inserted by Corpus Clean it is likely to be strongly biased towards this tool.

5.1.1.1 Corpus Preparation

Recall from Section 2.2.2 that WeScience uses a format with one sentence per line and that each line starts with a sentence identifier. To obtain a gold-standard for sentence segmentation we created a markup-free version of WeScience by removing the sentence identifiers and parsing each line as a separate page. Each of these tiny trees were then traversed as outlined in Section 3.4 where all nodes ware either `removed` (no markup is emitted, but any content is included) or `purged` (no text is emitted for the node and its children). Template inclusions were treated according to the rules developed in Section 2.1.3. Since Corpus Clean strips out all but a few templates, our template handling can be summarised (using the terms introduced in Section 3.3.3) by stating that we used the `remove` action for "IPA notice" (that produces a visual box with links to pages detailing the phonetic alphabet) and the `keep` action for others. This is different form the corpus preparation in our earlier experiments where all the remaining templates were `expanded`. Finally, any newlines introduced by our system were removed and empty strings were discarded.

Ideally we would have created an unsegmented pure-text version of WeScience by using a similar approach as that described above. Unfortunately, paragraph breaks are not included in WeScience. Since they are strong indicators of a sentence boundaries, not taking them into account in our experiments would yield an artificially high error rate. We have instead created the unsegmented version from the original article source.

When we created the gold standard we had to do some minor adjustments: We removed the `<blockquote>` tags at the beginning of WeScience sentences 10800560, 10800680 and 10800760 as they caused parse errors in the immediately following headings. These errors caused our tool to ignore those sections when creating the unsegmented set from the original article source. The wiki markup directive for heading (balanced pairs of equals signs) must be placed at the beginning of a line to have any effect.[1]

Unlike the preparation done by Read et al. (2012), code listings that spanned several lines (i.e. the contents of `<code>`-tags) were not included. The reason for discarding them was that since they would not be included in our corpus, we were not interested in how the different systems coped with their content.

[1]HTML-style headings (`<h1>`, `<h2>`, etc.) do not have this restriction.

5.1.2 Results and Discussion

We used character level F_1-score (described in Section 4.5.2.1) where the target class were sentence boundaries as the performance measure. This enabled us to take all the gold standard sentence breaks into account instead of only those that occurred directly after a limited set of punctuation characters. This is similar to the experiments in Read et al. (2012).

Since the unsegmented and gold standard segmented versions of the test corpus ware generated from different sources they were not perfectly aligned. The mis-alignments appeared to be the result of differences in our system and Corpus Clean and some manual corrections done in the creation of the corpus. The tool-induced differences were mostly due to inconsistencies in the template handling done by Corpus Clean where some template inclusions were removed for templates that were otherwise kept. Some HTML and XML tags that were escaped by `<nowiki>`-tags were also missing.

In order to cope with these differences we created a scoring script that that had a recovery mechanism that it used when the files it was comparing got out of sync. This recovery was done by looking ahead until it found a span of text that matched both files and continuing the scoring from that position. Sentence breaks that occurred inside of text that were only in one of the files were not taken into account when calculation the F_1-score.

We created a baseline by running the tools on a pure-text version of WeScience without doing any pre- or post-processing beyond ensuring that headings were followed by a single newline and that paragraph were followed by two consecutive newlines.

In the next run we aimed to get the best performance possible out of the tools by strategically forcing sentence breaks and by rewriting quote characters. Sentence boundaries was inserted before and after each block element as in most cases a disruption of the text flow is indicative of a sentence break. This was also done if for elements that we would not include in the corpus (for instance tables and horizontal lines). For most of the tools forcing sentence breaks was done by inserting paragraph breaks (i.e. double newlines). LingPipe, CoreNLP and MxTerminator do not interpret empty lines as the indication of a new paragraph (Read et al., 2012), so we forced sentence breaks by splitting the corpus into separate files for these tools.

We rewrote quote characters for the tools that benefited from this in our previous experiments (Read et al., 2012). The two rewriting schemes were: (1) “ASCII”, which meant replacing unicode quotes (code points `U+2018`, `U+2019`, `U+201C` and `U+201D`) with their ASCII counterparts (`'` and `"`). And (2) “LaTeX”, which meant using LaTeX-style directional quotes (``` `` ``` and `''`).

Table 5.3: Performance of sentence boundary detectors

	Baseline	Forced SB			
System	F_1-score	Quotes	F_1-score	Skipped	Sec
OpenNLP	0.9258		0.9796	493	1.6
GATE	0.9135		0.9681	2856	21.3
LingPipe (medical)	0.8998		0.9802	530	1.2
LingPipe (general)	0.9033	ASCII	0.9813	497	1.1
CoreNLP	0.8404	LaTeX	0.9692	499	1.5
MxTerminator	0.8396		0.9703	496	1.4
Punkt	0.9168		0.9709	495	11.1
Splitta	0.8794	ASCII	0.9894	534	16.3
Tokenizer	**0.9383**		**0.9931**	494	**0.4**
RASP	0.9324	ASCII	0.9903	542	**0.4**

The results from our experiments are displayed in Table 5.3. The F_1 measure for each tool with minimal preprocessing is shown below the heading "Baseline", while the score achieved when forcing sentence breaks are shown under the heading "Forced SB". The "Quote" column indicates the quote rewriting scheme used. As can be seen from the results, quote rewriting and forcing sentence boundaries increases the performance for all tools.

The column "Sec" displays the time in seconds each system needed to process the WeScience corpus as one single file and the number of characters skipped by the scoring-script is shown in the "Skipped" column. For most tools, the variations in this column mostly stems from different amounts of white-spaces in the out-of-sync text. The GATE sentence splitter stands out in this respect as the scoring script had to skip more than five times as many characters as for any other tool. This is because GATE occasionally leaves a few characters out (typically numerals and quotes) from the sentence annotations which effectively makes them disappear.

As stated above in Section 5.1.1.1, we also tested the performance of Corpus Clean. It gets an F_1-score of 0.9977, which is far more impressive than the preliminary error rate of approximately 4% reported by Ytrestøl et al. (2009, p. 190). It should be noted that the gold-standard sentence boundaries in WeScience was created by manually adjusting the sentence breaks identified by Corpus Clean. This means that it is likely that our gold standard has a considerable bias towards it. We will examine some of the procedures used by Corpus Clean to improve the sentence segmentation

Table 5.4: Effect of different ad-hoc rules

	Rule	TP	FP	FN	F_1-score
1	Boundaries around block elements	13876	127	67	0.9931
2	No newlines in <math>-tags	13876	72	67	0.9950
3	Splice enumerations	13876	62	67	0.9954
4	Single-sentence <code> and <math>	13877	55	66	0.9957
5	Single-sentence pre-formatted blocks	13876	39	67	0.9962
6	Single-sentence inline nodes	13540	34	403	0.9841
	Corpus Clean	13892	18	46	0.9977

to see if the are transferable to our system below in Section 5.2.

The best performing candidate, tokenizer, is the same system used by Corpus Clean in the creation of the WeScience corpus. This is likely to give it an advantage as the sentence boundaries inserted by it were in a way presented as the default choice for the annotators. Still, we use tokenizer in out system due to its high score and quick execution time. In our earlier experiments it was the highest scoring system on the all corpora except for the GENIA Corpus, which makes us believe that its performance is not a result of it being used in the creation of the gold standard.

5.2 Fine Tuning

Having picked a sentence segmenter for our system we did an error analysis in order to find ways of increasing its performance. Table 5.4 shows the cumulative effect of applying different heuristic rules. Corpus Clean is included for comparison. The first rule of forcing sentence breaks around block elements is the same approach described above. We use the Rules 1 to 4 in our system.

The Mediawiki extension "Math" renders "math-mode" LaTeX statements that are contained by <math>-tags. Since the markup for equations can become fairly long and complex, Wikipedia authors often use newlines to make it more readable. These newlines have no effect on how the equation is displayed. Removing all newlines from math statements (Rule 2) before passing the text to tokenizer greatly reduced the number of false positives from 127 to 72.

Even though wiki markup has facilities for creating numbered list, it is not uncommon to use other markup directives and starting each list element

with a number to create something with a similar appearance. Tokenizer tends to insert sentence boundaries after numbers followed by a full stop. Rule 3 is to remove sentence breaks after sentences that only consists of a number and a full stop.

Rules 4 and 5 are to remove any sentence breaks inserted inside `<code>`-tags, `<math>`-tags and pre-formatted text. While this bears some resemblance to Rule 2, it is more aggressive as newlines inside of `<code>` and pre-formatted blocks do have an effect on the visual appearance of the article. It also means that we are removing all sentence breaks inserted by tokenizer inside these markup elements. For pre-formatted text, this also carries the risk of overriding "visual sentence breaks" (i.e. newlines) intentionally placed there by the Wikipedia authors. This works well on WeScience that has a lot of articles with a technical subject matter where pre-formatted text is often used for pseudo code, makeshift diagrams and so on. We are however unsure of how well this rule generalises for the rest of Wikipedia so we do not make use of it in our system.

The final rule we tested was Rule 6, which is to remove any sentence breaks inside of links and markup directives for changing the appearance of text that are shorter than 35 characters. It was based on the intuition that these elements rarely cross sentence boundaries. Inter-article links have previously been used for recognising named entities on Wikipedia (Nothman, 2008), and it seems sensible to not allow sentence breaks within those. Links and inline style directives can also match grammatical constituents (Spitkovsky et al., 2010) which should not be split between sentences. Although in both of the aforementioned efforts the sentence segmentation was done as a preprocessing step and markup that spanned a sentence boundary was either discarded or split in two.

While this rule does result in a slight reduction of the number of false positives, it creates enough false negatives to lower the F_1-score. Most of the errors introduced by this rule fell into one of two categories: The first one being the way we disallowed sentence boundaries by bracketing the content of a markup element with markers (`<___` and `___>`) before processing the text with tokenizer. We then removed the markers and any newlines between them afterwards. A downside to this approach was that when used on the first, capitalised, word of a sentence it would "hide" the capitalisation from tokenizer and preventing it from inserting a sentence break. The other source of errors was that there were sentences that either ended inside of a directive or that were completely engulfed by one. Take for instance this sentence from the article "Artificial

intelligence":[2] `* The [[artificial brain]] argument: ''The brain can be simulated.''`. Tokenizer would correctly add a sentence boundary immediately after the full stop, which Rule 6 would remove as it cleared all sentence breaks inside of style directives. Errors in both categories can be reduced by finding another method of disallowing sentence breaks and by making an exception for markup elements ending in a punctuation mark. We did not pursue this further due to time constrains and the fairly low potential gain of only five less false positives.

As can be seen in Table 5.4, with an F_1-score of 0.9977 Corpus Clean still has an edge on our system on the WeScience corpus. While there is still room for improvements of the sentence segmentation in our system there are some methods employed by Corpus Clean that we will not adapt. Like for instance forcing sentence boundaries after the string ".org. " and years (i.e. two or four consecutive digits) followed by a full stop. We feel those address issues that are specific to the articles in WeScience and we are unsure of how well they will generalise to the rest of Wikipedia.

5.3 Restoring Markup

After first using tokenizer to detect the sentence boundaries in a markup-free string representation of an article section. We must then combine its output with the parse tree representation used internally in our system in order to create sentences that are marked up with GML.

This is done in two phases. The first phase is shown in Listing 5.1 and consists of traversing the parse tree and constructing a sequence of tokens. The definition for this token type is seen in lines 1-4. A token has two attributes: A reference to a node in the parse tree and type that is either `'start'` or `'end'`. The function for traversing the parse tree (`build_tokens()`) takes a tree and a reference to a sequence of tokens as parameters. If the node is a simple text leaf, it is added to the sequence and no new recursive calls are made (lines 7-9). For any other node types a start token (line 11) is added. The function then call itself recursively for any child nodes (lines 13-15) before adding a end token (line 19). Some nodes carry their textual content in one of several possible properties that is extracted by the function `getDisplayText()` that is called in line 17.

The algorithm for merging the pure text strings generated from tokenizer with the token sequence constructed by `build_tokens()` is displayed in Listing 5.2. The functions `getMarkupStart()` and `getMarkupEnd()` return

[2] two consecutive straight quotes is the wiki markup directive for rendering text in italics.

Listing 5.1: `build_tokens()`

```
class Token:
   def __init__(self, node, type):
      self.node = node
      self.type = type

def build_tokens(node, tokens):
   if isinstance(node, basestring):
      tokens.append(node)
      return

   tokens.append(Token(node, 'start'))

   if node.children:
      for c in tree.children:
         build_tokens(c, tokens)
   else:
      tokens.append(getDisplayText(node))

   tokens.append(Token(node, 'end'))
```

the opening and closing parts of a GML-tag for a given node. For example if the parameter is an "italics text style" node, these functions return the strings ⌊/¦ and ¦/⌋. The values (`PURGE`, `KEEP`, `REMOVE` and `ERSATZ`) that are stored in the `action` property of the nodes represent the actions described in Section 2.1.2. To recap them briefly:

Purge means that the node is not included in the output.

Keep means that the markup for both the node and its children is included.

Remove means that no markup is created for the node itself, but that its children are included as normal.

Ersatz means that the node is replaced with a self-closing tag.

For each sentence a number of tokens are iterated over by the inner loop starting at line 11. Markup for each sentence is accumulated in the `markup` variable, which is added to the list `res` each time the inner loop terminates (line 43). If the `skip_until` variable has been set by a starting node with the action **purge** or **replace**, no markup is produced until the corresponding end token is found (lines 14-15). If the token is a string, it is appended to the `markup` string (lines 19-29).

Listing 5.2: markup_sentences()

```
def markup_sentences(sentences, tokens):
    token_fragment = 0
    token_i = 0
    res = []

    for sentence in sentences:
        skip_until = None
        closing = False
        markup = ''

        while token_i < len(tokens):
          token = tokens[token_i]

          if skip_until:
            if skip_until == token.node:
                skip_util = None
            continue

          elif isinstance(token, basestring):
            nodetext = token[token_frag:]
            sentence_frag, node_frag = matchstring(sentence,
                nodetext)
            markup = sentence[sentence_frag:]
            sentence = sentence[sentence_frag:]
            if not sentence.strip():
                closing = True
            else:
                token_frag += node_frag
                break
          elif token.type == 'start' and closing:
            if not token.node.action == REDUCE:
                break
          elif node.action == PURGE and token.type == 'start':
            skip_until = token.node
          elif node.action == KEEP:
            if token.type == 'start':
                markup += getMarkupStart(token.node)
            else:
                markup += getMarkupEnd(token.node)
          elif node.action == ERSATZ and token.type == 'start':
            markup += getMarkupStart(token.node)
            skip_until = token.node
          elif node.action == REMOVE:
            pass

          token_i += 1

        res.append(markup)
    return res
```

The markup for most nodes is appended to `markup` as they appear in the sequence except when `closing` is set to `True`. This happens when all of the current sentence has been copied over to `markup` (lines 24-25). When `closing` is `True` only closing tags and tags for nodes that are **ersatzed** are added, while start tokens cause the inner loop to exit (lines 29-31). This means that as many closing tags are included at the end of a sentence as possible. Or to put it another way, a sentence will never begin with one or more closing tags.

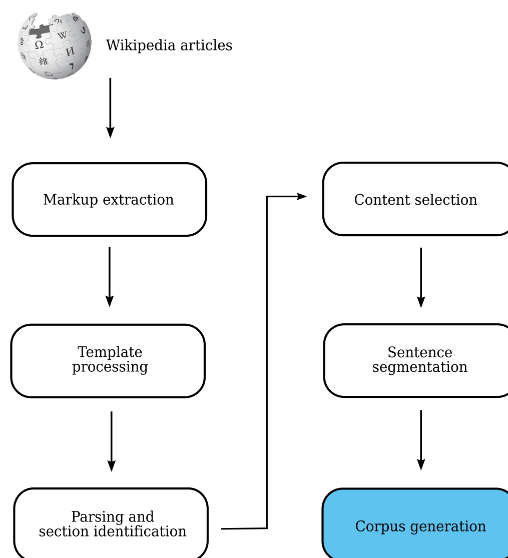
The case where a node with the **ersatz** action appears between sentences is ambiguous, as that node has not been seen by the sentence segmentor. The motivation for appending them to the last sentence is that it prevents sentences from being marked up like this: `⌊img⌋¦p⌋⌊p¦Foo bar.` Which would be interpreted as "inline image", "end paragraph", "start paragraph" followed by the text "Foo Bar". We will return to the details of GML in Section 6.1 and the full list of markup directives is provided in Appendix A.

Chapter 6

The Corpus

In this chapter we will describe our corpus, which we have named "WikiWoods 2.0". Section 6.1 details the Grammatical Markup Language (GML) that we use in our corpus. Section 6.2 describes its organisation. The final step of our system (sketched in Figure 6.1) which is "Corpus generation". An extrinsic evaluation and a comparison with the first release of the WikiWoods corpus is done in Section 6.3.

Figure 6.1: Overview of our system



Listing 6.1: The first few sentences from the article "Advanced Encryption Standard" marked up with GML.

```
1 ⌊document¦Advanced Encryption Standard¦document⌋
2 ⌊p¦In ⌊>¦cryptography¦>⌋, the ⌊*¦Advanced Encryption
     Standard¦*⌋ (⌊*¦AES¦*⌋), also known as ⌊*¦Rijndael¦*⌋,
     is a ⌊>¦block cipher¦>⌋ adopted as an ⌊>¦encryption¦>⌋
     standard by the ⌊>¦U.S. government¦>⌋.
3 It has been analyzed extensively and is now used worldwide,
     as was the case with its predecessor, the ⌊>¦Data
     Encryption Standard¦>⌋ (DES).
4 AES was announced by ⌊>¦National Institute of Standards and
     Technology¦>⌋ (NIST) as U.S. ⌊>¦FIPS¦>⌋ PUB 197 (FIPS
     197) on ⌊>¦November 26¦>⌋ ⌊>¦2001¦>⌋ after a 5-year
     standardization process in which fifteen competing
     designs were presented and evaluated before Rijndael was
     selected as the most suitable (see ⌊>¦Advanced
     Encryption Standard process¦>⌋ for more details).
5 [...]¦p⌋
```

6.1 GML

The work in defining GML was started by Flickinger et al. (2010), but this is the first complete description of the language. GML is intended to serve as an abstraction layer between other markup languages and NLP tools while at the same being easily parsable for both human and machine readers. It is capable of representing those markup directives that can have a linguistic significance, i.e. impact grammatical analysis, found in several other markup languages like wiki markup, HTML, and LaTeX. The existence of conversion software from these languages to GML would make it possible to use GML as an intermediate format for markup-aware NLP tools. The conversion software for one of these languages, viz. wiki markup, is presented in this work.

Human readability is accommodated by making GML a low-verbosity language: Most markup elements are named by a single character and there are only three "magical" characters:

Left delimiter ⌊ (`LEFT FLOOR`, `U+230A`)

Right delimiter ⌋ (`RIGHT FLOOR`, `U+230B`)

Middle delimiter ¦ (`BROKEN BAR`, `U+00A6`)

They are chosen because their appearance is distinctly different from the most other commonly used characters and because they see relatively little

use, i.e. usage of GML does not introduce a need for escaping of any "standard" characters, as for example in XML. Like XML and HTML, GML uses tags to assign properties to portions of text. Opening tags in GML begin with the left delimiter, which is followed by the name of a markup element and end with the middle delimiter. The corresponding closing tag begins with the middle delimiter followed (arguably redundantly) by the element name and is terminated by the right delimiter. For instance, in the second line in Listing 6.1 the element "bold" (`*`) is used on the abbreviation "AES" by surrounding it with an opening (`⌊*¦`) and a closing tag (`¦*⌋`).

In the spirit of human readability most markup directives consist of a single character and most of these have a glyph that is visually distinct from the rest of the text. Listing 6.1 shows an example taken from the first paragraph of the article "Advanced Encryption Standard". On the first line we see the name of the article surrounded by `document` tags. This element signifies the beginning of a new source document and doubles as a top-level heading. As with HTML tags, GML tags can nest. An example of this is shown in line 2, which starts with an opening paragraph tag (`⌊p¦`) that continues through the rest of this listing and contains several other markup elements. Several link elements (`>`) also appear in this markup sample.

For elements where we are certain that they never contain any relevant linguistic content but that we still want to keep, we **ersatz** (recall the four possible actions from Section 3.4) them with a self closing tag. An empty GML-tag starts with the left delimiter that is followed by the element name and the right delimiter. Currently the only type of element for which we use this action are inline images (`img`), for which the self closing tag is `⌊img⌋`.

It is possible to assign additional properties to a markup element by adding attributes to the closing tag. One element that uses attributes is headings, where the attribute is its level. The GML-representation of a second-level heading "Notes" looks like this `⌊=¦Notes¦2¦=⌋`. Storing this information in the closing tag might seem surprising, at first, but has the benefit of ensuring that the actual content of an element always appears directly after the first middle delimiter.

Since we hope to see GML used on other sources of user generated content, there are no restrictions on how the tags are nested. It is for instance permissible for GML-tags to partially overlap like this `⌊*¦bold ⌊/¦and¦*⌋ italics¦/⌋`. This makes it possible to represent markup that is either written "sloppily" or in a language that does not have any restrictions on proper nesting, i.e. strict tree structure of markup elements, without normalising it. The way templates function makes wiki markup one of the languages where proper nesting is not guaranteed.

While the three control characters used in GML do not occur in most

text, they do occasionally do. They can be escaped by surrounding them with the left and right delimiter like this:

Escaped LEFT FLOOR ⌊⌊⌋

Escaped RIGHT FLOOR ⌊⌋⌋

Escaped BROKEN BAR ⌊¦⌋

6.1.1 Markup Elements

Some markup directives have been introduced in Section 6.1 above and there is a complete list of the markup elements in wiki markup and GML in Appendix A. This section contains the motivation for why certain elements were included in the corpus while others were not.

Tables Recall from Section 2.1.2 that tables are not included in the corpus even though it is not uncommon for tables to contain longer spans of running text. This is because of the interaction between row and column headings and the content of a table cell which makes it difficult to interpret this text.

Images Most of the images are discarded with their caption as mentioned in Section 2.1.2. This means that we may occasional lose sentences with relevant linguistic content, but as with the textual content of table cells the captions often refer to the content of its image in a way that is difficult to process correctly. Images that appear in the normal text flow often take the role of a sentence constituents and removing those would lead to a sentence potentially becoming ungrammatical. Inline images are **ersatzed** with an image-tag (⌊`img`⌋).

Links and formatting We include links and most of the markup directives for changing the appearance of text, as there are several possible ways to take advantage of these in NLP. Inter-article links have been used for named entity recognition (Nothman, 2008) and both links and text formatting directives can be indicative of sentence constituents (Spitkovsky et al., 2010). Flickinger et al. (2010) point out that italics can indicates a use-mention distinction or text in a foreign language.

Structural elements Some markup elements are used to indicate article structure. Perhaps the most prominent ones are headings (`document` and `=`) and paragraphs (`p`). These might not always be interesting for a researchers that are examining linguistic phenomena, their presence

makes it easier for a human to make sense of an article. Since these elements are identified by the surrounding tags little effort is needed to ignore them should it be necessary.

Semantic tags Some markup elements do not explicitly declare how text should be rendered, but rather how their content should be interpreted. Examples of such markup elements in Wikipedia include `<cite>`, `<var>` and `<abbr>` that are used to signify that some text is a citation, a variable or an abbreviation. It seems likely that the presence of these markup elements can contribute to downstream processing and we include all such elements in our corpus.

Templates Our treatment of templates is detailed in Section 3.3. Those templates that we feel add value to the corpus are included with the GML element `x`. For instance the template inclusion `{{Convert|62|kg|lb|abbr=on}}` is represented like this: `⌊x¦62 kg (140 lb)¦Convert¦62¦kg¦lb¦abbr=on¦x⌋`. As always in GML, the text intended for display (i.e. the expanded form) comes immediately after the opening tag. The name of the template and its invocation parameters are included as attributes on the closing tag.

Code listings and equations As with images, we discard code listings that introduces a visual break the normal text flow (i.e. `<source>`-tags). Math elements (`<math>`) and source code snippets (`<code>`) that appear in the normal text flow are, as with images, often sentence constituents that should not be removed. We do however include the contents of these elements in order to make the corpus more readable for humans — on the assumption that inline code segments tend to be very short.

6.2 Corpus Generation and Structure

The final stage of our system is "Corpus generation". Recall from Section 5.3 that the last that was done in the previous stage, "Sentence segmentation", was to combine the sentence breaks found by a text-only tool (tokenizer) with the tree representation used internally by our system. This process resulted in a list of GML marked up strings, where each string is one separate sentence. Before the corpus is generated the articles are sorted alphabetically, and superfluous white space characters are removed.

Our corpus is organised in the same way as WeScience and Wiki-Woods 1.0. It is organised into segments which each hold 100 articles,

Figure 6.2: Article and Sentence identifiers

```
 Art. Id  S. Id
[1000100000010] | document ¦ ...
 P            D
```

where each sentence is on a single line. Each of the segments has a unique five digit-identifier, with the lowest identifier being 00101.

Figure 6.2 shows how a sentence will appear in our corpus. Each line start with the article and sentence identifiers which are enclosed by square brackets, and are followed by a horizontal bar separating it from the sentence. The sentence identifier is a seven digit number, where the first digit (marked "P") is a placeholder that is always set to one. We have set aside the first hundred article identifiers for the articles in the WeScience corpus. Sentence identifiers are unique within an article and are globally unique when combined with an article identifier. The last digit (marked "D") in the sentence identifier defaults to zero in order leave room for manual corrections of the sentence segmentation (Ytrestøl, 2009; Flickinger et al., 2010).

6.3 Evaluation and Comparison with WikiWoods 1.0

Table 6.1 displays some figures on the original release of WikiWoods and our corpus (WikiWoods 2.0), which contains almost twice as many articles as WikiWoods 1.0. This difference is likely the result of the different approaches to article selection taken by Corpus Clean and our system. Only articles of at least 2,000 characters are included in WikiWoods 1.0, whereas in our work we included all articles that had at least one section that made it through the "Content selection" stage (detailed in Chapter 4). In terms of the number of sentences and word-tokens, the corpora are roughly the same size. That indicates that our system discards far more content than Corpus Clean from most articles. And that this is replaced with content from short articles.

The average sentence length is longer in WikiWoods 2.0 than in WikiWoods 1.0. One would expect that sections with a high level of noise would on average contain shorter sentences than clean sections. As noise often appears in the form of lists of links, reference lists (after the removal of "Cite"-templates) and so on, which usually consist of short sentences (often a single word). In order to quantify this, we collected the sections that

Table 6.1: Figures from WikiWoods 1.0 and WikiWoods 2.0

	WikiWoods 1.0	WikiWoods 2.0
Number of articles	1,289,187	2,451,272
Number of sentences	55,221,131	51,169,566
Number of tokens[a]	843,215,993	850,972,241
Avg. sentence length	15.27	16.63

[a] Tokenized by white space.

Listing 6.2: The short article “"J" Is for Judgment” as it appears in WikiWoods 2.0.

```
[1000013900000] |⌊document¦"J" Is for Judgment¦document⌋
[1000013900010] |⌊p¦⌊*¦⌊/¦"J" Is for Judgment¦/⌋¦*⌋ is the
   tenth novel in ⌊>¦Sue Grafton¦>⌋'s "Alphabet" series of
   mystery novels and features ⌊>¦Kinsey Millhone¦>⌋, a
   private eye based in ⌊>¦Santa Teresa, California¦>⌋.¦p⌋
```

would normally be discarded in the “Content selection” stage from 10,000 randomly chosen articles. This resulted in a collection of a little more than 18,000 “dirty” sections of about 92,000 sentences. The average sentence length for these sections were about five, which is far lower than for both the corpora. We interpret this as an indication of our corpus containing less noise, which is often manifests itself as enumerations and other very short sentences, and more full sentences of grammatical English.

As mentioned above, we do not place a minimum length restriction or any other arbitrary limits on articles and have instead opted to include all articles that have at least one clean section. The reason for this is twofold: First, we believe that short articles, even those containing just a heading and a single sentence, often contain relevant linguistic content. Listing 6.2 shows one such single-sentence article, which contains grammatical English that describes the article subject. This is also true of disambiguation pages, that is pages that links to articles detailing different meanings of the same term, which might seem to be an obvious source of noise. Take for instance the article “Giant Steps (disambiguation)” shown in Listing 6.3, it starts with a short definition and continues with a list of other possible meanings.[1]

[1]Admittedly, the content in disambiguation pages is often redundant as the different meanings of the word are usually more thoroughly explained in the linked articles. But this is not always the case: In the article in Listing 6.3 the last link does not point to an

Listing 6.3: The short article “Giant Steps (disambiguation)” as it appears in WikiWoods 2.0.

```
[1084145300000] |⌊document¦Giant Steps
   (disambiguation)¦document⌋
[1084145300010] |⌊p¦⌊*¦⌊/¦⌊>¦Giant Steps¦>⌋¦/⌋¦*⌋ is a 1960
   album by jazz musician ⌊>¦John Coltrane¦>⌋.¦p⌋
[1084145300020] |⌊p¦⌊*¦Giant Steps¦*⌋ may also refer to:¦p⌋
[1084145300030] |⌊p¦⌊●¦⌊#¦⌊>¦Giant Steps (band)¦>⌋, dance
   pop duo from England that consisted of vocalist Colin
   Campsie and bassist/keyboardist George McFarlane¦#⌋
[1084145300040] |⌊#¦⌊>¦⌊/¦Giant Steps¦/⌋ (⌊/¦Boo Radleys¦/⌋
   album)¦>⌋, the third album by ⌊>¦The Boo Radleys¦>⌋¦#⌋
[1084145300050] |⌊#¦⌊>¦⌊/¦Giant Steps¦/⌋ (book)¦>⌋,
   autobiography of ⌊>¦Kareem Abdul-Jabbar¦>⌋, which he
   co-authored with Peter Knobler¦#⌋
[1084145300060] |⌊#¦⌊>¦"Giant Steps" (composition)¦>⌋, the
   first track on the album of the same name by John
   Coltrane¦#⌋
[1084145300070] |⌊#¦⌊>¦⌊/¦Giant Steps¦/⌋ (Gentle Giant
   album)¦>⌋, a compilation album by ⌊>¦Gentle
   Giant¦>⌋¦#⌋¦●⌋¦p⌋
```

While short pages contribute a great deal of relevant linguistic content to WikiWoods 2.0 there might be use-cases where they are not as interesting as longer articles. Someone who is aiming to perform a discourse level analysis on Wikipedia articles or wishes to study rhetorical structure might want to discard all articles of less than a certain threshold of sentences. Due to our line-based and readable format, this should be an straightforward task.

The other reason for not having any set criteria for including articles is that we already have a well-functioning classifier for identifying non-relevant content. As detailed in Section 4.1, an approach using machine learning (such as ours) is often able to generalise from the samples seen during training. Whereas an approach using hand-written rules might be limited to the types of noise seen by its author.

6.3.1 Extrinsic Evaluation

In order to determine if the proportion of relevant linguistic content is larger in our corpus than WikiWoods 1.0 we measured the parsability of our corpora using the LinGO English Resource Grammar (ERG) (Flickinger,

existing article and the removal of this disambiguation page could potentially erase that knowledge from the corpus.

Table 6.2: Figures from the samples from WikiWoods 1.0 and WikiWoods 2.0

	WikiWoods 1.0	WikiWoods 2.0
Number of articles	130,000	129,469
Number of sentences	5,076,028	4,259,116
Number of tokens[a]	82,232,331	76,637,757
Avg. sentence length	16.20	17.99

[a] Tokenized by white space.

2000) against earlier results on the original WikiWoods. Before we discuss the results from the parsing we will spend a few paragraphs on the preparation and composition of the samples.

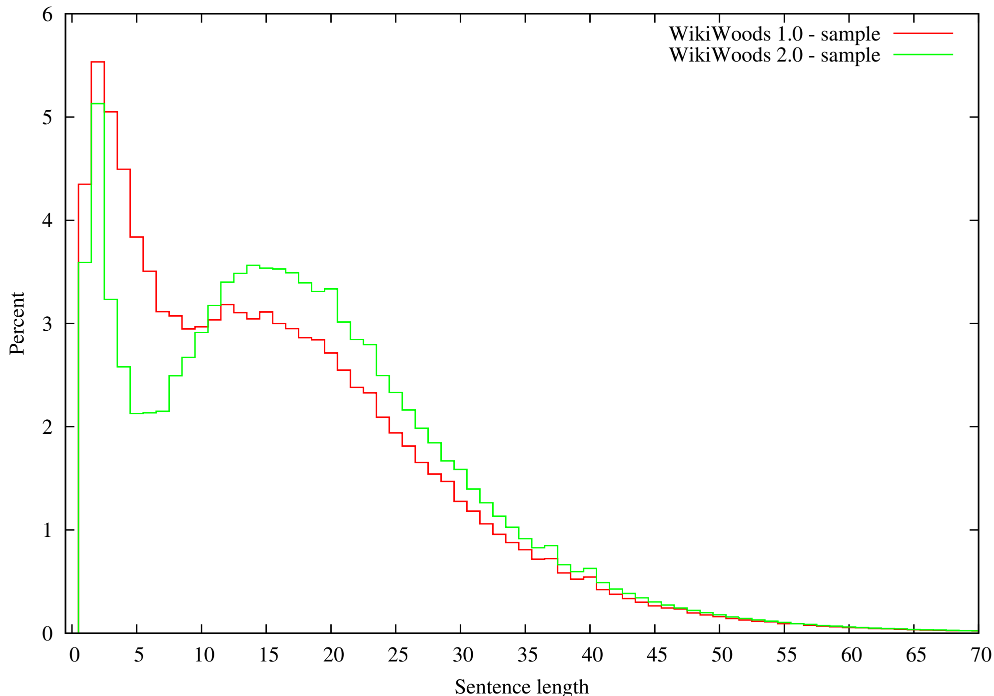
The articles in the first edition of WikiWoods are organised in segments. Each segment consists of 100 articles and have a unique five-digit identifier. We collected a sample of 1,300 segments from WikiWoods 1.0. We selected segments with six as the middle digit in their identifier, which amounts to selecting one hundred consecutive segments out of every thousand in order to get a stratified sample. The articles articles corresponding to these segments in WikiWoods 1.0 in their "raw" form (each article in a separate file containing the article name and the wiki markup) were then processed by our system creating GML representations of the articles.

The figures for the samples corresponding to those shown in Table 6.1 are displayed in Table 6.2. Slightly more than 500 articles got completely removed by our classifier, so the samples contain are about the same size in that regard. But when looking at the sentence and token count it becomes apparent that the classifiers have removed a substantial amount of content. For WikiWoods 2.0 as a whole the added content from the shorter articles replaces the content discarded in the "Content selection"-phase, but since there are no short articles in the segments from WikiWoods 1.0 the size of the WikiWoods 2.0 sample is smaller in terms of tokens and sentences.

As with the corpora as a whole, the average sentence length is longer in our corpus than in the original release of WikiWoods. As stated above we take this to mean that WikiWoods 2.0 is a "cleaner" corpus as noise often consists of short sentences. Figure 6.3 shows the distribution of sentences by length in the two samples. This plot shows how the cleaning performed by our system have produced a sample with a proportionally larger number of long sentences. The distribution of sentences longer than 15 words are almost identical in the two samples.

A GML pre-processing module was added to the ERG, which exploits

Figure 6.3: Sentence distribution by length



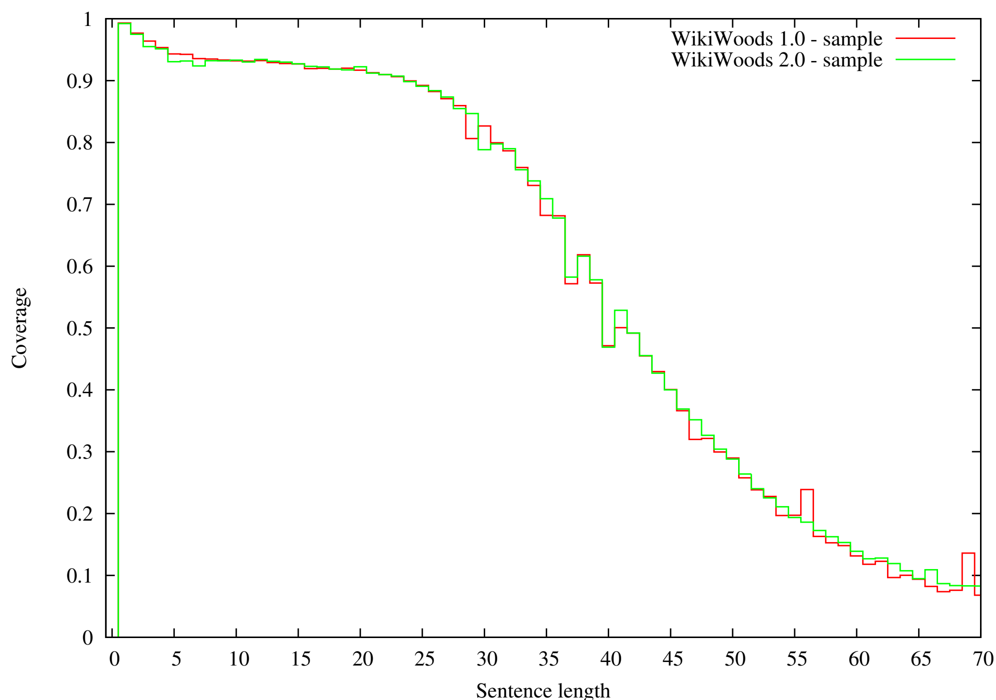
markup in essentially the same manner as the existing Wiki module: Math and code elements are replaced with placeholders, the same is done for templates indicating a foreign language. With the exception of the pre-processing modules, the setups were identical for parsing both samples.[2]

The ERG is able to parse 88.38% of the sentences in WikiWoods 1.0, while the coverage on our corpus is slightly lower: 87.37% where about seven percent of the failures are due to resource exhaustion and morphology and lexical look-up errors. This slight drop in coverage might seem surprising at first. But recall from Figure 6.3 that our system removes a large number of short sentences (consisting of 10 or fewer words), which in effect shifts the distribution of the sample towards longer sentences. Since longer sentences are more difficult to parse than short ones, the fact that the coverage is about the same for both samples seems to indicate that the content of WikiWoods 2.0 is of a higher quality.

A plot showing the percentage of successful parses by sentence length for both samples is displayed in Figure 6.4. Notice how the coverage for each

[2]I am indebted to Stephan Oepen for his effort in adapting the ERG to GML and for parsing the samples.

Figure 6.4: Parsing coverage by sentence length



sentence length is almost identical. Together with the proportionally higher number of "long" sentences in the sample from WikiWoods 2.0 (shown in Figure 6.3) this indicates that longer sentences stands for a larger percentage of successful parses in WikiWoods 2.0.

Chapter 7

Conclusion

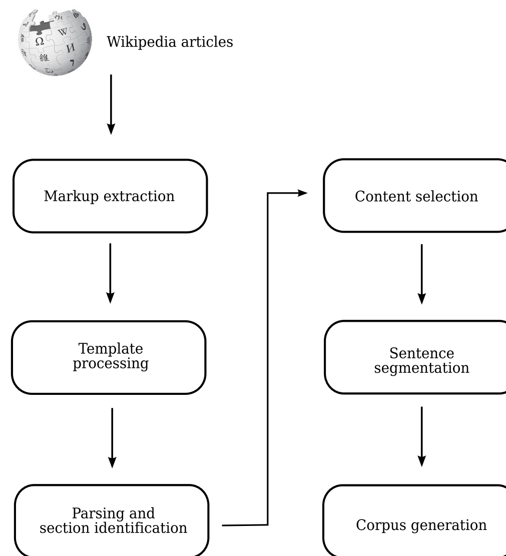
We have presented a system for generating large corpora with high-quality linguistic content from Wikipedia, and other Mediawiki wikis, and have demonstrated its capabilities by creating a corpus from a database snapshot. Both our system (Wikipedia Corpus Builder) and the corpus (WikiWoods 2.0) are available publicly.[1] As stated in Chapter 1, some of the qualities that give Wikipedia the potential to be an very important resource for both NLP and IR are that it contains a large number of articles and that these are generally well-written.

Chapter 2 gave an overview of how the content on Wikipedia is organised and introduced wiki markup, which is the language that its pages are written in. It further detailed how some of the directives in wiki markup, namely those from the extension ParserFunctions and templates, behave in a way that makes it best viewed as a general programming language. These elements must be properly interpreted in order to separate genuine, authored content from boilerplate and ensuring the correct interpretation of other markup elements. The final part of that chapter was used to examine earlier work on Wikipedia from an NLP standpoint as well as some of the available tools for parsing wiki markup. We finally gave an architectural sketch of our corpus generating system.

In Chapter 3 we performed a feasibility study of using either the Mediawiki engine or mwlib as the basis for the markup processing components of our system. This study resulted in our decision to use mwlib. We then detailed the three first stages of our system (sketched in Figure 7.1): "Markup extraction", "Template processing" and "Parsing and purification". Out of these three, the "Template processing" stage was given most attention due to the complex nature of templates and the fact

[1] At `http://github.com/larsjsol/wcb` and `http://www.delph-in.net/wikiwoods/`

Figure 7.1: Overview of our system



that they can generate both noise and extra information that can be useful for downstream users. We also created a modified version of the template sub-system in mwlib in order to increase performance (in terms of speed) and to implement the different ways we handle template inclusions.

The "Content selection" stage was detailed in Chapter 4. This chapter provided background on some of the earlier efforts on page cleaning. Our approach of using two character-level n-gram models to classify article sections as "clean" or "dirty" was presented along with our in-depth experiments on the performance of different classifier configurations. Our system uses add-one smoothed models with an n-gram order of four. The last part of that chapter was dedicated to some of the more difficult cases encountered by the annotators in the creation of the gold standard test set and how they led to a more refined notion of what should constitute relevant linguistic content.

The "Sentence segmentation" stage was detailed in Chapter 5, which started with empirical tests of how well several existing tools perform. This is one of several NLP tasks that benefits from the presence of markup and we demonstrated how relatively simple measures resulted in noticeable improvements. The latter part of Chapter 5 detailed how we combine the sentence breaks introduced by a text-only tool with the internal representation of articles used by our system.

Chapter 6 gave an account of GML, which is an abstract and simple markup language that is designed for NLP tasks and that aims to strike a

balance between human and machine readability. It further detailed the organisation of our corpus and showed an extrinsic evaluation done by parsing a subset from both our corpus and the original WikiWoods 1.0 corpus (Flickinger et al., 2010).

Our most important results can be summarised as follows:

- We have created very large corpus that we have made publicly available. The combined effect of our two methods of noise removal, selectively removing templates and markup elements and discarding "dirty" sections, made this corpus cleaner than its predecessor. This was shown by a quabtitative comparison to WikiWoods 1.0 by the results from a limited extrinsic evaluation where a subset of WikiWoods and our corpus was parsed.

- We have described a method of creating high-quality corpora from collections of user generated content and developed an implementation if this method that operates on database snapshots from wikis running Mediawiki. Our implementation is freely available.

- Proper template handling is necessary for doing anything but shallow processing of wiki markup. Furthermore, the majority of template inclusions are of a relatively small number of templates, and this makes it possible to affect a large number of inclusions by changing how a relatively small number of templates are expanded.

- Our approach of using character level n-gram models for classifying content works well on article sections. The best performing configuration achieved an F_1-score in excess of 0.95 on the "silver" standard, which is the test set which is likely to have a similar composition as Wikipedia as a whole. Add-one smoothed n-gram models generally outperformed the other, more sophisticated, smoothing algorithms used in our experiments.

- The performance of sentence boundary detectors can be greatly increased by taking markup elements into account. Not doing so leads to poor performance as many of the sentence breaks are often indicated solely by markup and lacks punctuation.

7.1 Future Work

Below are some brief mentions of how the work in this thesis can be improved or extended upon.

Streamlining our System While our system can be used on other Mediawiki snapshots than the one used in this work without modification, we can not claim that it is user friendly. For instance, some of the auxiliary scripts used to tune language models and examine the distribution of template inclusions make some assumptions about specific file names. In order to mitigate this we will work on improving the parameterisation system itself, as well provide technical documentation. Our system is available for download at `http://github.com/larsjsol/wcb`.

Marking up WeScience with GML The WeScience corpus (Ytrestøl et al., 2009) was created from the same snapshot as our corpus and has gold-standard sentence segmentation. We have not included the articles in WeScience in the initial release of our corpus as we plan to create a GML marked up variant of it.

Technically this could be done by using the text only variant of WeScience that we created for evaluating the sentence segmentors (detailed in Section 5.1.1.1) in place of tokenizer and merging the sentence breaks with the parse trees from the same articles (this method is detailed in Section 5.3). Since we are able to parse the markup directly from WeScience, few manual corrections should be necessary.

Markup-Aware Sentence Segmentation As shown in Chapter 5, sentence segmentation benefits from markup awareness. However, our current approach of forcing sentence breaks between block elements still leave something to be desired as it breaks apart those sentences that do span such markup elements. The sentences that are broken up by our approach are often those that contain enumerations that are marked up as lists.

One possible way of taking advantage of markup is to disallow sentence breaks within certain markup elements. We tested this assumption in one of our experiments (Rule six in Section 5.2), but due to implementation details in our setup we were unable to properly measure the effect of this approach. We wish to revisit this method in the future.

It would also be interesting to see if tags like `<abbr>` (signifying that its content is an abbreviation) could be used to disambiguate full stops followed by a capitalised word. This tag is only used twice in the WeScience corpus (Ytrestøl et al., 2009), which is not enough to perform experiments on. Unfortunately the use of this tag seems to be fairly sporadic so it might only offer a marginal benefit in most texts.

Content Selection With a Finer Granularity Since sentences can span several markup elements, our classifier operates on articles sections. While these usually have similar characteristics from beginning to end there are some that contain both relevant linguistic content and noise. With our current approach we include such sections, but finding a smaller unit for the content selection would make it possible for our system to derive even cleaner corpora than it does now.

Our current approach also leaves out image captions and textual content in table cells. A part of developing a more fine-grained content selection could be to find ways of extracting those instances that can be useful even without the image itself or the table structure.

Glossary

article The encyclopedic articles in Wikipedia. These reside in the main namespace.

block element Markup element that disrupts or appears outside of the normal text flow. For instance tables, paragraphs and various lists..

boilerplate Often repeated text like navigational elements and copyright notices, we aim to remove all boilerplate along with other types of noise.

Canola Web corpus created with the KrdWrd framework (Steger and Stemle, 2009).

CleanEval A shared task held in 2007 on removing boilerplate and discovering document structure in web pages (Baroni et al., 2008).

Collection Extension for Mediawiki that adds functionality to export articles in various document formats and order printed copies.

Corpus Clean Pipeline used to create the WeScience and WikiWoods corpora from a Wikipedia snapshot (Ytrestøl, 2009; Ytrestøl et al., 2009; Flickinger et al., 2010).

dirt *see* noise

disambiguation page Page that links to articles on the different meanings of a term.

extension A plug in for Mediawiki. Most offer functionality that has no bearing for our project, but some such as Math and ParserFunctions adds new directives to the wiki markup.

flatten The second phase of mwlib's template expansion, this is where the evaluation of template inclusions take place.

GML Grammatical Markup Language, a markup language that covers a subset of wiki markup, HTML and LaTeX. It is designed to balance human- and machine readability in corpora.

information box A visual box with short labelled statements summarising the article, noise for our purpose.

inline element Markup element that appears within the normal text flow. Links and text formatting are common elements if this kind..

jusText Algorithm for boilerplate removal with a heuristic approach (Pomikálek, 2011).

KrdWrd A framework for annotating noise and clean text in web pages (Steger and Stemle, 2009).

L3S-GN1 A manually annotated data set created from news articles linked from Google News, page content is marked as headline, full-text, boilerplate, etc. (Kohlschütter et al., 2010).

Mediawiki Software that is used to run Wikipedia and other wikis.

message box A visual box usually shown at the top of an article containing meta information. A message box is shown in Figure 1.1.

navigation box A visual box with links to other pages with related topics, noise for our purpose. An information box is shown in Figure 1.1.

NCLEANER A boilerplate removal tool, uses two character level n-gram models to classify spans of text either clean or dirty (Evert, 2008).

noise Content we do not want in the resulting corpus, like meta information, bibliographies and so on. The content *do* want is relevant linguistic content.

page Any page on Wikipedia, pages in the main namespaces are usually referred to as articles.

ParserFunctions A Mediawiki extension that adds support for flow control and basic mathematical operations.

redirect A page that forwards the visitor to another page.

relevant linguistic content Article content in the form of natural language that contains some information about the world. The type of content we want to include in our corpus.

SRILM the SRI Language Modelling Toolkit, contains utilities for building and running n-gram models (Stolcke et al., 2011). We use it to create a classifier for clean and dirty text.

template Page intended to be included in an article or other templates, some of them make use of the more advanced features of wiki markup and their behavior resembles that of programs.

template markup We use this term for the directives of wiki markup that is evaluated during template expansion.

WeScience Corpus containing 100 Wikipedia articles in the NLP domain, created with Corpus Clean (Ytrestøl, 2009).

wiki Website where visitors can create and modify pages, the most notable being Wikipedia.

wiki markup Markup language used to format pages on Wikipedia.

Wikimedia Foundation The organisation that runs Wikipedia and several other wikis.

WikiWoods Corpus containing most of the articles from a Wikipedia snapshot, created with Corpus Clean (Flickinger et al., 2010).

Bibliography

Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives (2007). Dbpedia: A nucleus for a web of open data. *The Semantic Web*, 722–735.

Baroni, M., F. Chantree, A. Kilgarriff, and S. Sharoff (2008). Cleaneval: a competition for cleaning web pages. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Morocco, pp. 638–643.

Bauer, D., J. Degen, X. Deng, P. Herger, J. Gasthaus, E. Giesbrecht, L. Jansen, C. Kahna, T. Kriiger, R. Martin, et al. (2007). FIASCO: Filtering the internet by automatic subtree classification, osnabruck. In *Building and Exploring Web Corpora: Proceedings of the 3rd Web as Corpus Workshop, incorporating Cleaneval*, Volume 4, Belgium, pp. 111–121.

Bird, S., E. Klein, and E. Loper (2009). *Natural Language Processing with Python*. Beijing: O'Reilly.

Briscoe, T., J. Carroll, and R. Watson (2006). The second release of the RASP system. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, Austrailia, pp. 77–80.

Chakrabarti, D., R. Kumar, and K. Punera (2007). Page-level template detection via isotonic smoothing. In *Proceedings of the 16th international conference on World Wide Web*, Canada, pp. 61–70.

Chen, S. F. and J. Goodman (1998). An empirical study of smoothing techniques for language modeling. Technical report, Harvard University, Computer Science Group.

Evert, S. (2007). StupidOS: A high-precision approach to boilerplate removal. In *Building and Exploring Web Corpora: Proceedings of the 3rd*

Web as Corpus Workshop, Incorporationg CleanEval (WAC3), Belgium, pp. 123–133.

Evert, S. (2008). A lightweight and efficient tool for cleaning web pages. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Morocco, pp. 3489–3493.

Flickinger, D. (2000). On building a more efficient grammar by exploiting types. *Natural Language Engineering 6*(1), 15–28.

Flickinger, D., S. Oepen, and G. Ytrestøl (2010). Wikiwoods: Syntacto-semantic annotation for English Wikipedia. In *Proceedings of the 7th International Conference on Language Resources and Evaluation*, Malta, pp. 1665–1671.

Francis, W. and H. Kucera (1982). Frequency analysis of English usage. *Boston, MA*.

Gillick, D. (2009). Sentence boundary detection and the problem with the US. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pp. 241–244.

Girardi, C. (2007). Htmcleaner: Extracting the relevant text from the web pages. In *Building and Exploring Web Corpora: Proceedings of the 3rd Web as Corpus Workshop, incorporating Cleaneval (WAC3)*, Volume 4, pp. 141–143.

Hofmann, K. and W. Weerkamp (2007). Web corpus cleaning using content and structure. In *Building and Exploring Web Corpora: Proceedings of the 3rd Web as Corpus Workshop, Incorporationg CleanEval (WAC3)*, Belgium, pp. 145–154.

Johnson, R. A. and G. K. Bhattacharyya (2010). *Statistics: Principles and Methods* (6th ed.). Hoboken, NJ, USA: John Wiley & Sons, Inc.

Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (2nd ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Kim, J., T. Ohta, Y. Tateisi, and J. Tsujii (2003). GENIA corpus – a semantically annotated corpus for bio-textmining. *Bioinformatics 19*(suppl 1), i180–i182.

Kiss, T. and J. Strunk (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics 32*(4), 485–525.

Kneser, R. and H. Ney (1995, may). Improved backing-off for M-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing, 1995. ICASSP-95.*, Volume 1, pp. 181 –184 vol.1.

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

Marcus, M., M. Marcinkiewicz, and B. Santorini (1993). Building a large annotated corpus of english: The Penn Treebank. *Computational linguistics 19*(2), 313–330.

Marek, M., P. Pecina, and M. Spousta (2007). Web Page Cleaning With Conditional Random Fields. In *Building and Exploring Web Corpora: Proceedings of the Fifth Web as Corpus Workshop, Incorporationg CleanEval (WAC3)*, Belgium, pp. 155–162.

Morante, R. and E. Blanco (2012). *SEM 2012 shared task: resolving the scope and focus of negation. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics - Volume 1: Proceedings of the main conference and the shared task*, SemEval '12, Stroudsburg, PA, USA, pp. 265–274. Association for Computational Linguistics.

Ney, H. and U. Essen (1991). On smoothing techniques for bigram-based natural language modelling. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91.*, Canada, pp. 825–828.

Ney, H., U. Essen, and R. Kneser (1994, January). On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language 8*(1), 1–38.

Nothman, J. (2008). *Learning Named Entity Recognition from Wikipedia.* Honours Bachelor thesis, The University of Sydney Australia.

Nunberg, G. (1990). *The Linguistics of Punctuation.* Number 18 in Lecture Notes. Stanford, CA: CSLI Publications.

Pomikálek, J. (2011). *Removing Boilerplate and Duplicate Content from Web Corpora.* Ph. D. thesis, Masaryk University.

Read, J., R. Dridan, S. Oepen, and L. J. Solberg (2012). Sentence Boundary Detection: A Long Solved Problem? Accepted for *The 24th International Conference on Computational Linguistics (Coling 2012)*. India.

Read, J., D. Flickinger, R. Dridan, S. Oepen, and L. Øvrelid (2012, May). The WeSearch Corpus, Treebank, and Treecache. A comprehensive sample of user-generated content. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC'12)*.

Reynar, J. C. and A. Ratnaparkhi (1997). A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied natural language processing (ANLP 97)*, USA, pp. 16–19.

Saralegi, X. and I. Leturia (2007). Kimatu, a tool for cleaning non-content text partsfrom HTML docs. In *Building and Exploring Web Corpora: Proceedings of the Fifth Web as Corpus Workshop, Incorporationg CleanEval (WAC3)*, Belgium, pp. 163–168.

Spitkovsky, V. I., D. Jurafsky, and H. Alshawi (2010). Profiting from markup: Hyper-text annotations for guided parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Sweden, pp. 1278–1287.

Steger, J. and E. Stemle (2009). KrdWrd, Architecture for Unified Processing of Web Content. In *Proceedings of the Fifth Web as Corpus Workshop (WAC5)*, Spain, pp. 63–70.

Stolcke, A., J. Zheng, W. Wang, and V. Abrash (2011). SRILM at sixteen: Update and outlook. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, USA.

Vatanen, T., J. J. Väyrynen, and S. Virpioja (2010). Language identification of short text segments with n-gram models. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation LREC'10*, Malta, pp. 3423–3430.

Weizheng, G. and T. Abou-Assaleh (2007). GenieKnows Web Page Cleaning System. In *Building and Exploring Web Corpora: Proceedings of the 3rd Web as Corpus Workshop, Incorporationg CleanEval (WAC3)*, Belgium, pp. 135–154.

Witten, I. and T. Bell (1991, July). The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory 37*(4), 1085 –1094.

Ytrestøl, G. (2009). Technical Summary-Selection and preprocessing of the WeScience corpus. Technical report, University of Oslo, Department of Informatics.

Ytrestøl, G., S. Oepen, and D. Flickinger (2009). Extracting and annotating Wikipedia sub-domains. In *Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories*, The Netherlands, pp. 185–197.

Appendix A

Elements of GML and Wiki Markup

Table A.1: Elements of wiki markup

Name	Action	GML element	GML tag
Section	keep	heading	⌊=¦text¦=⌋
Article	keep	document / heading	⌊document¦text¦document⌋
Source	purge		
Code	keep	Source code	⌊f¦text¦f⌋
BreakingReturn	purge		
HorizontalRule	purge		
Index	purge		
Teletyped	keep	tele typed	⌊t¦text¦t⌋
Reference	purge		
ReferenceList	purge		
Gallery	purge		
Center	remove		
Div	remove		
Span	remove		
Font	remove		
Strike	keep	strike through	⌊-¦text¦-⌋
ImageMap	purge		
Ruby	remove		
RubyBase	remove		
RubyText	remove		

Continues on the next page...

Table A.1: Elements of wiki markup

Name	Action	GML element	GML tag
RubyParantheses	purge		
Deleted	keep	strike through	⌊-¦text¦-⌋
Inserted	keep	underline	⌊_¦text¦_⌋
Caption	purge		
Table	purge		
Row	purge		
Cell	purge		
Abbreviation	keep	abbreviation	⌊.¦abbrevation¦extended term (optional)¦.⌋
Math	keep	code	⌊f¦text¦f⌋
DefinitionList	keep	definition list	⌊:¦list items¦:⌋
Item	keep	list item	⌊#¦text¦#⌋
ItemList	keep	list	⌊•¦list items¦•⌋
DefinitionTerm	keep	term	⌊:¦term¦:⌋
DefinitionDescription	keep	indent	⌊↦¦description¦↦⌋
Heading	keep	heading	⌊=¦text¦=⌋
TimeLine	purge		
Italic	keep	italic	⌊/¦text¦/⌋
Bold	keep	bold	⌊*¦text¦*⌋
Strong	keep	bold	⌊*¦text¦*⌋
Blockquote	keep	quote	⌊"¦text¦"⌋
Underline	keep	underline	⌊_¦text¦_⌋
Overline	keep	strike through	⌊-¦text¦-⌋
Sub	keep	subscript	⌊,¦text¦,⌋
Sup	keep	superscript	⌊^¦text¦^⌋
Small	keep	small	⌊↓¦text¦↓⌋
Big	keep	big	⌊↑¦text¦↑⌋
Cite	keep	citation	⌊cite¦text¦cite⌋
Var	keep	code	⌊f¦text¦f⌋
Preformatted	keep	pre formatted	⌊pre¦text¦pre⌋
Poem	keep	pre formatted	⌊pre¦text¦pre⌋
Comment	purge		
URL	keep	link	⌊>¦text¦>⌋
ArticleLink	keep	link	⌊>¦text¦>⌋
InterwikiLink	keep	link	⌊>¦text¦>⌋
CategoryLink	purge		

Continues on the next page...

Table A.1: Elements of wiki markup

Name	Action	GML element	GML tag
ImageLink	replace	image	⌊img⌋
LangLink	purge		
NamedURL	keep	link	⌊>¦text¦>⌋
NameSpaceLink	keep	link	⌊>¦text¦>⌋
Paragraph	keep	paragraph	⌊p¦text¦p⌋
Emphasized	keep	italics	⌊/¦text¦/⌋
Hieroglyphs	purge		
Template	n/a	template	⌊x¦expansion¦name¦arg1 (optional)¦...¦x⌋

Appendix B

Template Lists

B.1 Most Used Templates

Table B.1: Most included templates

Direct	Total	Name	Description	Action
1256248	1375733	Flagicon	Displays a small flag	remove
520927	520982	Reflist	Displays a list of references.	remove
409021	415170	Cite web	Creates a reference	See ˆCite
227473	227578	Fact	"[Citation Needed]"	See Fix
216303	217974	Cite journal	Creates a reference	remove

Continues on the next page...

Table B.1: Most included templates

Direct	Total	Name	Description	Action
198974	207888	Convert	Converts between units	See / ˆconvert/
186172	275796	Mp	{{Mp|a|b}} -> a<sub>b</ sub>	expand
185066	203211	Cite book	Creates a reference	remove
129606	421673	Flag	Displays a small flag next to a link to a nation.	expand (the flag icon will dissapear anyways)
129244	130316	Succession box	Creates a “succession box”	expand
125704	125791	Main	“Main article: Foo”	remove
124516	130955	End	Table end “|}”	expand
123163	123168	Unreferenced	“This article does not cite any sources...”	See Ambox
116178	153727	Coord	Creates a link to geohack based on coordinates.	keep
115083	115139	Nihongo	Indicates that this text is in Japanese	See ˆNihongo
113672	113673	GR	Creates a reference	See ˆCite
110500	110562	Birth date and age	Displayes a data followed by the number of years since then	keep
104045	109539	S-start	Start of a succesion box	expand
102915	104580	0	Invisible zero	remove
102196	102321	Cite news	Creates a reference	See ˆCite
99930	99932	Disambig	“This disambiguation page lists articles...”	remove
95106	113402	IPA	Indicates that this text is in IPA notation	See ˆIPA

Continues on the next page...

Table B.1: Most included templates

<table>
<tr><th>Direct</th><th>Total</th><th>Name</th><th>Description</th><th>Action</th></tr>
<tr><td>93414</td><td>93464</td><td>Football squad player</td><td>Creates table cells with infomation about a player</td><td>expand</td></tr>
<tr><td>87902</td><td>87902</td><td>Taxobox</td><td>Creates a information box, does not use {{Infobox}}</td><td>remove</td></tr>
<tr><td>80336</td><td>81842</td><td>Election box candidate with party link</td><td>Creates table cells with infomation about a politician</td><td>expand</td></tr>
<tr><td>76187</td><td>76622</td><td>Infobox Settlement</td><td>Expands into a table containg an {{Infobox}}</td><td>remove</td></tr>
<tr><td>73915</td><td>73915</td><td>FlagIOCathlete</td><td>Displays a small flag next to the name of an athlet and a link to “nation at the XXXX olympics”.</td><td>keep</td></tr>
<tr><td>72549</td><td>73243</td><td>Infobox Album</td><td>Expands into a table containg an {{Infobox}}</td><td>remove</td></tr>
<tr><td>67406</td><td>67406</td><td>Ushr</td><td>Returns a link to a US congressional district</td><td>expand</td></tr>
<tr><td>64025</td><td>65774</td><td>Fb</td><td>Retuns a smal flag next to a link to a national football team</td><td>expand (the flag icon will dissapear anyways)</td></tr>
<tr><td>62641</td><td>96140</td><td>Coor dms</td><td>Creates a link to geohack based on coordinates.</td><td>See /^coor/</td></tr>
<tr><td>62241</td><td>62841</td><td>Yes</td><td>Creates a coloured table cell</td><td>expand</td></tr>
<tr><td>57871</td><td>57926</td><td>Baseball Year</td><td>{{Baseball Year|1990}} -> [[1990 in baseball|1990]]</td><td>expand</td></tr>
<tr><td>55589</td><td>64412</td><td>Coor d</td><td>Creates a link to geohack based on coordinates.</td><td>See /^coor/</td></tr>
<tr><td>50949</td><td>55960</td><td>Flagcountry</td><td>Displays the flag and the name of a country</td><td>expand</td></tr>
</table>

Continues on the next page. . .

Table B.1: Most included templates

Direct	Total	Name	Description	Action
50288	50288	Gbmappingsmall	Expands to {{Gbmaprim}}	See Gbmaprim
48344	48344	Imdb name	Creates a link to imdb	expand
46998	47694	No	Creates a coloured table cell	expand
44991	45082	USA	Expands to {{flag|United States}}	See Flag
42427	42429	Refimprove	This article needs additional citations...	See Ambox
42401	42415	Rating-5	Displayes an icon of stars followed by the text X/ 5 stars.	keep
41392	41392	Sortname	Creates a link that is sortable	expand
40994	43106	Citation	Creates a reference	remove
40314	41836	Flagathlete	Displayes a flag, name of an athlete and a country code.	expand
39451	39451	Goal	Displayes a soccerball icon followed by the time it was scored.	keep
39044	39044	Imdb title	Creatse a link to imdb	expand
39009	39028	Dts2	Creates a html span	expand
38863	130821	Lang	Indicates that this text belongs to a particular language	keep
36381	36713	Episode list	Creates a table of summaries of episodes	remove
35939	35955	For	"For other uses see..."	See Dablink
33883	33883	Infobox Football biography	Displayes an infobox.	remove
33497	33499	Coor title dms	Creates a link the article Coordinates followed by a set of coordinates.	See /^coor/
33118	33118	Infobox Film	Displayes an infobox.	remove

Continues on the next page...

Table B.1: Most included templates

Direct	Total	Name	Description	Action
32678	32678	Infobox Musical artist	Displayes an infobox.	remove
32256	47567	Party shading/ Democratic	Returns a css style	expand
31200	31376	Commons cat	Creates a box with a link to a category in Wikimedia Commons	Se Commons
31088	31088	Coor title dm	Creates a link the article Coordinates followed by a set of coordinates.	See /^coor/
30266	37941	Nts	Creates a html span	expand
29408	60867	Commons	Creates a box with a link to a article in Wikimedia Commons	remove
29004	29004	Jct	Displayes an icon followed by a junction of American highways	expand(or do the same as with flag)
28866	28866	Orphan	This article is an orphan, as few or no other articles link to it...	See Ambox
27712	27712	Footballbox	Creates an information box	remove
27653	27656	Birth date	Creates links to to year and date, used to indicate that this text is a date.	See {{Birth date and age}}
27580	27793	Election box end	Table end "|}"	expand

Continues on the next page...

Table B.1: Most included templates

Direct	Total	Name	Description	Action
27218	39181	Party shading/ Republican	Returns a css style	expand
27123	27123	National football squad player	Retuns table cells	expand
27035	27035	Lifetime	Creates category links to birth and death year	expand
26995	26996	Cleanup	This article may require cleanup to...	See Ambox
26935	35762	Portal	Creates a box with a link to a portal page	remove
26845	27124	See also	See also: Article and Article	remove
26063	26065	Death date and age	Death date and age	See {{Birth date and age}}
25981	26175	Election box begin	Expands to the start of a table	expand
25906	25906	Persondata	Creates a table with biographical data, used to aid automatic information gethering	remove
25439	59360	Coor dm	Returns a URL	See / ˆcoor/
24722	24722	Infobox Single	Creates a information box	remove
23998	24220	En icon		See Languageicon

Continues on the next page. . .

Table B.1: Most included templates

<table>
<tr><th>Direct</th><th>Total</th><th>Name</th><th>Description</th><th>Action</th></tr>
<tr><td>23588</td><td>23588</td><td>Height</td><td>Converts between units of length</td><td>Do the same as with Convert</td></tr>
<tr><td>23276</td><td>170007</td><td>S-ttl</td><td>Creates a “succession box”</td><td>expand</td></tr>
<tr><td>22963</td><td>22964</td><td>Expand</td><td>Please help improve this article or section by expanding it.</td><td>See Ambox</td></tr>
<tr><td>22834</td><td>22834</td><td>Infobox Actor</td><td>Creates an information box</td><td>remove</td></tr>
<tr><td>22692</td><td>22692</td><td>Fb r</td><td>Creates a table cell with background colour (Fb r == Football result)</td><td>expand</td></tr>
<tr><td>22553</td><td>22553</td><td>Mapit-US-cityscale</td><td>Finds the coordinates of a city based on the page name</td><td>expand (might break wikimarkup if removed)</td></tr>
<tr><td>21778</td><td>22425</td><td>Fr icon</td><td></td><td>See Languageicon</td></tr>
<tr><td>21576</td><td>22214</td><td>-</td><td>Expands to <br clear="all" / ></td><td>expand</td></tr>
<tr><td>21247</td><td>23223</td><td>Col-end</td><td>Expands to <p></ p> |}</td><td>expand</td></tr>
<tr><td>21019</td><td>21019</td><td>MedalGold</td><td>Expands to table cells and table end</td><td>expand</td></tr>
<tr><td>21009</td><td>21009</td><td>MedalSport</td><td>Expands to table cells and table end</td><td>expand</td></tr>
<tr><td>20739</td><td>20739</td><td>MedalBottom</td><td>Expands to |}</td><td>expand</td></tr>
<tr><td>20680</td><td>20874</td><td>Election box turnout</td><td>Expands into table cells</td><td>expand</td></tr>
</table>

Continues on the next page…

Table B.1: Most included templates

Direct	Total	Name	Description	Action
20147	20445	Cquote	Quotes text	keep
20140	20149	Refbegin	Starts a list of references	expand
20126	20135	Refend	Ends a list of references	expand
20072	22477	Col-begin	Returns table start	expand
19901	22360	Player	Displayes a player followed by {{flagicon}}	expand
19767	20081	Sort	Returns a html span	expand
19639	159347	S-bef	Creates a cell in a succession box	expand
19527	31007	NRHP color	returns a style attribute	expand
19412	19575	Ref	Creates a footnote that (its body can be displayed by {{Note}})	remove
19336	24546	Unicode	Indicates that this text is in unicode	expand
19313	19313	Ja-stalink	Creates a link to a Japanese train station	expand
19164	153268	S-aft	Creates a cell in a succession box	expand
13119	377464	Nowrap	Returns a html span-tag with the “nobreak” style.	keep
12330	129229	Dablink	Creates a disambiguation link “See also ...”	remove
9713	342628	·	Expands to “ ”’·””’	expand
5602	490710	!	Returns “|”	expand
2608	113179	Age	Takes a date and returns the number of years since then.	keep
2566	64271	Click	Creates a clickable image	remove
725	139282	Languageicon	Creates part of a link	expand
611	60392	Oscoor	Like {{coor}} but uses “Ordnance Survey grid reference”	keep
472	833358	Navbox	Displays a navigation box	remove

Continues on the next page...

Table B.1: Most included templates

Direct	Total	Name	Description	Action
306	54351	Gbmaprim	Expands to {{Oscoor}}	See Oscoor
255	78940	Infobox	Creates an “information box”	remove
174	446475	Ambox	Displays a “message box” (e.g. This article may require cleanup...)	remove
166	249318	Fix	Creates a inline message in square brackets	remove
121	51766	Navbox Musical artist	Creates a navigation box	See Navbox
82	95953	Tnavbar	Creates part of a navigation box	expand
45	60797	LangWithName	Expands to [[X language: {{lang|...]]	expand
43	43528	Getalias	Expands to {{Country data X}}	expand
27	851822	Taxobox colour	Returns a html colour code.	expand
12	421676	Country flag2	Displays a small flag next to a link to a nation.	expand
11	69139	Country data Germany	Expands to an filename of an image of a flag.	Se Flagicon
10	59080	Asbox	“This article is a stub. You can help by expanding it.”	remove
3	97058	Country data France	Expands to an filename of an image of a flag.	Se Flagicon
3	47307	Convert/ km	Expanded by {{Convert}}	Se Convert
2	1375664	Country flagicon2	Displays a small flag	remove
2	232284	Coor URL	Createa a link to geohack, used by {{coord}}	remove
2	59865	Convert/ ft	Expanded by {{Convert}}	Se Convert

Continues on the next page...

Table B.1: Most included templates

Direct	Total	Name	Description	Action
1	278781	Country data United States	Expands to an filename of an image of a flag.	Se Flagicon
1	73244	Infobox Album/ link	Creates a link to an album	expand
1	64665	Convert/ m	Expanded by {{Convert}}	Se Convert
1	56037	Country data Canada	Expands to an filename of an image of a flag.	Se Flagicon
1	55958	Country flag-country2	Used by {{Flagcountry}}	expand
1	55545	Coord/ display/ title	Creates a link the article Coordinates followed by a set of coordinates.	expand
1	52823	US county navigation box	Creates a navigation box	See Navbox
0	210002	BS-overlap	Displayes an overlapping icon (of a German rail line?)	expand, might break syntax otherwise
0	170231	Convert/ LoffAonSoff	Expanded by {{Convert}}	Se Convert
0	159195	Country flaglink	Shows an icon of a flag followed by “Flag of Foo”	expand
0	132660	Tracklist/ Track	Creates table cells	expand

Continues on the next page. . .

Table B.1: Most included templates

Direct	Total	Name	Description	Action
0	107723	Chembox entry	Creates a {{Chembox Foo}} if it is supplied with a parameter...	remove
0	104802	1x	Returns its argument unchanged	expand
0	101636	Convert/ round	Expanded by {{Convert}}	Se Convert
0	91404	Convert/ pround	Expanded by {{Convert}}	Se Convert
0	90845	Country data Italy	Expands to an filename of an image of a flag.	Se Flagicon
0	90168	Convert/ LoffAoffDbSoff	Expanded by {{Convert}}	Se Convert
0	79062	Country data Australia	Expands to an filename of an image of a flag.	Se Flagicon
0	77433	WPMILHIST Infobox style	Expands to css style info	expand
0	67750	Country data England	Expands to an filename of an image of a flag.	Se Flagicon
0	61926	Convert/ LoffAonDbSoff	Expanded by {{Convert}}	Se Convert
0	59897	Country data Spain	Expands to an filename of an image of a flag.	Se Flagicon
0	54618	Country data Japan	Expands to an filename of an image of a flag.	Se Flagicon

Continues on the next page…

Table B.1: Most included templates

Direct	Total	Name	Description	Action
0	54297	Coord/ display/ inline	Returns its argument unchanged	See / ˆcoor/
0	53543	Country data United Kingdom	Expands to an filename of an image of a flag.	Se Flagicon
0	44498	Convert/ mi	Expanded by {{Convert}}	Se Convert
0	43880	Coord/ display/ inline,title	displayes coordinates followed a link to [[Geographic coordinate system]]	See /ˆcoor/
0	43528	Country get alias2	Used by {{Getalias}}	expand
245	245	IPA notice	Displays a notice about the IPA notation.	remove
42	42	Javadoc:EE	Creates a link to the Java documentation	keep
471	471	Javadoc:SE	Creates a link to the Java documentation	keep
896	896	Harvard citation text	Inline citation	keep
4732	4732	Harvard citation	Inline citation	keep
6627	6627	Harvard citation no brackets	Inline citation	keep
175	175	Harvard citations	Multiple inline citations	keep
2348	13960	Transl	Transliteration	keep
8517	9069	IAST	Transliteration of sanskript	expand
6	6	IAST1	Transliteration of sanskript	expand

Continues on the next page. . .

Table B.1: Most included templates

Direct	Total	Name	Description	Action
3696	4199	Audio	Creates a link to an audio file inside a pair of parathesis	remove
192	192	CoorOS	“The geographic coordinates are from the [[Ordnance Survey]].”	remove
364	364	ThisDateInRecentYears	A box with links to “date articles”	remove

B.2 Template Naming Conventions

Table B.2: Template naming conventions

Inclusions	Regex	Description	Action
1268560	stub$	This article is a stub...	remove
581244	^infobox	Information boxes	remove
137442	party	Does not match a uniform class of templates	expand
93215	icon$	icons, with and without text	expand
68355	^lang	The slightly different pattern /^Lang-/ (this exludes {{Lang Son Province}}) mathces templates that all designate that some text is of a specified language, most of these expands to {{Lang}} via {{LangWithName}}.	keep
63412	^football	Does not match a uniform class of templates	expand
54706	county	Navigation boxes for American counties	remove
53255	^col	Helper templates for building tables, removing them would break the wiki markup syntax.	expand
50451	^election	Does not match a uniform class of templates	expand
44901	squad	Does not match a uniform class of templates	expand
38508	list$	Does not match a uniform class of templates	expand
28958	^cite	Create citations.	remove
28316	^pbb	PBB = Protein Box Bot, information boxes about mammalian proteins.	remove
26971	line$	Navigaion boxes for subway/rail/bus lines	remove
26351	entry$	Most of these are used as part of information/navigation boxes	expand
26278	^afl	Does not match a uniform class of templates	expand
23226	start$	Many of these are the start of tables or divs	expand
22252	color$	HTML colour codes.	expand
19598	^chembox	Information boxes about chemicals, many of these take other chembox templates as arguments.	expand
19432	^auto	The content of these are a quantity of a measuring unit (except {{Auto isbn}}	keep
19146	name$	Does not match a uniform class of templates	expand
18836	label$	Does not match a uniform class of templates	expand
18637	district	Does not match a uniform class of templates	expand

Continues on the next page...

Table B.2: Template naming conventions

Inclusions	Regex	Description	Action
18610	^harvard	Does not match a uniform class of templates (but the Harvard citation variants are kept)	expand
18214	^canadian	Does not match a uniform class of templates	expand
18125	box$	Does not match a uniform class of templates	expand
17374	end$	Ends tables or divs.	expand
17240	^expand	Resembles stub (but only 5 templates match).	remove
17197	squad$	Does not match a uniform class of templates	expand
16521	^french	Does not match a uniform class of templates	expand
15221	^party	Does not match a uniform class of templates	expand
14849	^can$	Matches {{Can}} and {{CAN}}, the first is a navigation box for a rock band and the second expands into a canadian flag followed by a link to the article "Canada".	expand
14417	^coor	{{Coor URL}} will be cought and kept.	keep
13487	team$	Does not match a uniform class of templates	expand
12679	radio$	Does not match a uniform class of templates	expand
12550	par$	Does not match a uniform class of templates	expand
12397	^chset	Does not match a uniform class of templates	expand
12375	communes$	Navigaion boxes for french communes.	remove
12297	link$	Most of these produce links.	expand
11715	^ipa	All except {{IPA-Notice}} displays text in IPA notation.	keep
11087	^geobox	Does not match a uniform class of templates	expand
10963	date$	Does not match a uniform class of templates	expand
10884	colour$	HTML colour codes.	expand
10828	^geolinks	Does not match a uniform class of templates	expand
10507	class	Does not match a uniform class of templates	expand
10144	^anime	Does not match a uniform class of templates	expand
9779	historic	Does not match a uniform class of templates	expand
9697	off$	Does not match a uniform class of templates	expand
9640	^politics	Navigation boxes for politics related articles.	remove
9180	with	Does not match a uniform class of templates	expand
198986	^convert	Converts between units	keep
118854	^nihongo	Indicates that this text is in Japanese	keep

Continues on the next page...

Table B.2: Template naming conventions

Inclusions	Regex	Description	Action
4960	calendar	Calendars	remove
8434	redirect	“Foo redirtects here...”	remove