
Aengus Walton
B.A. (Mod.) Computer Science, Linguistics and a Language
Final Year Project, May 2009
Supervisor: Carl Vogel
Second Reader: Martin Emms
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

__________________________  May 8, 2009

Aengus Walton
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

__________________________________________ May 8, 2009
Aengus Walton
Acknowledgements

I would like to thank Carl Vogel for his guidance and seemingly inexhaustible reservoir of novel ideas while doing this project, and also for his support and sound advice throughout my entire time at University.

Thanks are also due to Evan Jones, for making his suite of Wikipedia plain-text extraction tools available on the web under a friendly license, and for his advice and insight on certain points.

I also wish to thank my family for their support during this and previous years, and for putting up with my distance and often narky mood in times of stress.
## CONTENTS

4.2 Segmentation of corpus and removal of wiki-markup .......................... 12  
4.3 Compiling Histories ......................................................... 15  

5 Compilation of statistics ..................................................... 17  
5.1 Primary Statistics ............................................................ 17  
5.2 Readability Scores ............................................................ 18  
5.3 Further Statistics ............................................................. 20  
5.4 Summary ................................................................. 23  

6 Randomized Corpus ......................................................... 25  
6.1 Purpose ................................................................. 25  
6.2 Preparation ................................................................. 26  
6.3 First Method: “Bags Approach” ............................................. 26  
6.4 Second Method .............................................................. 27  

7 Analysis of results ............................................................. 28  
7.1 Analysis of preliminary results .............................................. 30  
7.2 Word-length frequency distributions ....................................... 31  
7.3 Sentence-length frequency distributions .................................... 32  
7.4 Word Counts ................................................................. 36  
7.5 Correlation between numbers of authors/edits, and lexical richness ...... 36  
7.6 Correlation between readability test scores and authors, edit counts ...... 37  
7.7 Standard deviation of sentence length ....................................... 38  
7.8 Wikipedia-specific Data ....................................................... 38  

8 Conclusion ................................................................. 40  
8.1 Future work ................................................................. 41
A List of Programs

A.1 Pre-processing ................................................. 46
  A.1.1 Subcorpus Filterers .................................... 46
  A.1.2 XML Dump Parser ...................................... 49
A.2 Corpus Analysis ............................................... 60
  A.2.1 Contributors Scraper ................................... 60
  A.2.2 Statistics Generator ................................... 64
  A.2.3 Word-length Frequency Generator ..................... 82
  A.2.4 Project Gutenberg Sentence-length Frequency Generator .......... 86
A.3 Randomized Corpus Generation .............................. 88
  A.3.1 Corpus Concatenator .................................. 88
  A.3.2 First Method .......................................... 90
  A.3.3 Second Method ....................................... 97
  A.3.4 Randomized Corpus Statistics Generator ............... 101
List of Figures

4.1 Rendered wiki-markup. ............................................................. 13

7.1 Word-length frequency distributions for the original corpora .......... 32
7.2 Sentence-length frequencies for the original corpora ..................... 33
7.3 Sentence-length frequencies for the RAND corpora ...................... 33
7.4 Sentence-length frequencies for the RAND2 corpora .................... 34
7.5 Sentence-length frequencies for some literary works .................... 34
7.6 Sentence-length standard deviations for original corpora ............... 38
7.7 Sentence-length standard deviations for RAND2 corpora ............... 38
List of Tables

5.1 Final size of subcorpora .................................................. 24

7.1 Average values per article per subcorpus, Part 1: Primary Statistics .......................... 29
7.2 Average values per article per subcorpus, Part 2: Readability Score Data .................. 29
7.3 Average values per article per subcorpus, Part 3: Wikipedia-specific Data
   (Randomized corpora omitted as this data is not applicable to them) ......................... 30
7.4 Correlation between $A_t$ and $W_t$, $E_t$ and $W_t$ ........................................ 36
7.5 Type-token ratio $((\frac{W_u}{W_t})_a)$; correlation between per-article author count and
   per-article edit count, and type-token ratio ......................................................... 37
7.6 Correlations between $A_t$, $E_t$ and the number of links and missing citation
   notices on an article and its number of translations ............................................. 39
Abstract

Wikipedia is a very popular, free encyclopedia, editable by anyone. As such, it presents interesting new opportunities and challenges to those involved in the fields of corpus linguistics, computational stylistics and stylometry. How does the collaboration of hundreds or thousands of different editors on a single article affect its style? With so many people working on the corpus as a whole, is it largely homogenous or not? Indeed, how can such characteristics be measured in such a special case as Wikipedia’s? This paper aims to address and answer questions of this kind.
“Style is the mind skating circles around itself as it moves forward”

Robert Frost

“If any man wish to write in a clear style, let him be first clear in his thoughts; and if any would write in a noble style, let him first possess a noble soul.”

Johann Wolfgang von Goethe
Chapter 1

Introduction

1.1 The Web as a corpus

The immense growth of the world-wide-web has provided many new possibilities for those in the field of computational linguistics. The use of the web as a corpus has become a busy field of research in corpus linguistics, with papers discussing all manner of related topics, from the technicalities – such as the dubious ownership of material on the web and copyright concerns that arise as a result – and logistics of the process [Cavaglia and Kilgarriff 2001] [Kilgarriff 2001], to the compilation of multi-lingual web-corpora [Baroni and Kilgarriff 2006], and the use of the web as a tool in word sense disambiguation [Agirre and Martinez 2000] or as a resource for descriptive semantic information [Fujii and Ishikawa 2000].

Since it is still quite young, Wikipedia has so far been slightly neglected as a resource for corpus linguistics, despite the free licensing of its content and the unique opportunities it presents. Those papers which have been written on it
mainly deal with criticism of the encyclopedia’s anarchic nature [Hu et al. 2007] and with the size and characteristics of the corpus [Denoyer and Gallinari 2006], with only a few researchers, notably Gabrilovich and Markovitch [2006] and Strube and Ponzetto [2006], using the Wikipedia corpus as a tool in linguistics research (as a means to achieve more accurate text categorisation and create models of semantic relatedness, respectively).

The aim of this project was to conduct a preliminary study of certain stylistic aspects of Wikipedia and ascertain just how homogenous the corpus is.

1.2 Stylistics and homogeneity

The use of the word “stylistics” in this paper really refers to the field of stylometry, that is, the application of stylistics to written language in order to to, for example, attribute authorship to anonymously written or disputed documents, or in the field of forensic linguistics.

The earliest known discussion of the aims and methods of this field are from Mendenhall [1887], who introduces the idea by paraphrasing Augustus DeMorgan (possibly his Budget of paradoxes), who writes, “some time somebody will institute a comparison among writers in regard to the average length of words used in composition, and that it may be found possible to identify the author of a book, a poem, or a play, in this way”.

With the advent of computers, computational stylistics has experienced a great amount of growth in recent times.

This paper will focus mainly on what effects, if any, the collaborative nature of Wikipedia has on its style and homogeneity.
Chapter 2

Wikipedia

2.1 Overview

Wikipedia, established in the beginning of 2001, is a “free encyclopedia which anyone can edit.” As of 2009, it is being actively developed by its users (so-called Wikipedians) in 236 languages, having experienced immense growth since 2002.

All textual material submitted to Wikipedia is released under the GNU Free Documentation License, a “copyleft” license for free documentation. Similar to the GNU General Public License for open-source software, it gives readers the rights to copy, redistribute and modify a work and requires all copies and derivatives to be available under the same license.

Database dumps of all Wikipedias are performed routinely and made available online\(^1\). The database dump used in this research was of the English Wikipedia and was performed on the 8th October, 2008\(^2\).

\(^1\)http://en.wikipedia.org/wiki/Wikipedia:Database_download
\(^2\)The timestamp on the database archive therefore being 20081008
2.2 The Corpus

The English Wikipedia corpus is immense: 2.8 million active contributors have created and collaborated on almost 3 million articles\(^3\). In addition to this, there are:

- **Talk Pages**: Pages which host discussion and debate amongst users about the content of each article. Discussions can, of course, range from anything as mundane as the aesthetic lay-out of an article, to lengthy, heated debates on the factuality of points in certain contentious articles, such as *George W. Bush* or *Noam Chomsky*.

- **User Pages**: Pages which act as a “profile page” for each user. Users tend to list their proficiency in languages, which subjects and articles they contribute to overwhelmingly, and which Wikipedias they edit.

- **Portal Pages**: Pages which act as “portals”, introducing visitors to key topics in subjects as broad as *Politics* and *Music* or as narrow as *Anarchism* and *Guitars*.

- **Wikipedia-specific Pages**: These pages contain much meta-information, such as guides on the editing of the encyclopedia, general style and etiquette, and how to treat copyrighted material.

- **Image Pages**: Image pages serve simply to display an image in full resolution and meta-data which may or may not have been annotated by the uploader.

\(^3\)These figures are correct as of 3rd May 2009.
Although there could be some very interesting research done regarding the content of Talk pages in relation to their article pages, in this project we focused solely on the article pages themselves.

The entire Wikipedia corpus, including all previous revisions of all pages (but excluding possibly-copyrighted material such as images, sound and video themselves) amounts to about 1 terabyte\(^4\) of XML. Wikipedia also distributes a dump consisting of simply the current revisions of articles which amounts to about 20 gigabytes of XML data. It was decided to use this dump for analysis, and to use a web tool to get a picture of each article’s revision history.

However, given the time-frame of the project and the constraints of hardware available, 20 gigabytes is still too much data to conduct a full analysis of. It was decided to compile subcorpora of this corpus. This process is discussed later, in section 4.1.

### 2.3 Edit process

One contributing factor to the immense size of Wikipedia is the fact that each time an edit is made to an article, an entirely new revision is stored. Along with each new revision, a record is made of the time and date of its submission and the user name of its author. Those who aren’t registered as Wikipedia users can also make edits – in this case, their IP address is stored as their identifier.

The user can also opt to write a short summary of the edit and why it was performed, and can mark the edit as a minor or major revision. In this project, these two characteristics were ignored.

\(^4\)One terabyte = 1000\(^4\) bytes
3.1 Python

Python is a high-level programming language which supports multiple programming paradigms (such as being object-oriented, imperative, functional). Developed by Guido van Rossum since the mid-90s, it is as well suited to building large-scale programs as it is to quickly solving problems with rough-and-ready scripts.

Python 2.5 (more specifically, the Enthought Python Distribution\(^1\) of it) was chosen as the primary language of this project for this reason, and also because of the wealth of libraries (such as BeautifulSoup and the NLTK) available for it.

\(^1\)EPD (www.enthought.com) is a Python distribution tailor-made to people who require scientific, mathematical or statistical libraries. It comes with libraries such as numpy and matplotlib, which the NLTK depends on. For some reason, their install procedures fail on a 64-bit Windows XP installation, but EPD has no problem installing and using these libraries on such an architecture. It was chosen and used for this reason.
3.1.1 BeautifulSoup

BeautifulSoup\(^2\) is a fantastic XML/HTML parser for Python. Its strengths are its indifference to the quality of the XML being parsed, full Unicode support and wide array of handy methods for navigating the parse tree. This library made programming tools to parse Wikipedia’s XML mark-up not only possible, but even enjoyable.

3.1.2 Python Natural Language Toolkit

The Python Natural Language Toolkit\(^3\) (NLTK) is an open-source toolkit for linguists and computer scientists designed to facilitate natural language processing. The project was begun at the University of Pennsylvania by Steven Bird and Edward Loper in 2001 and is now actively developed by many contributors across the globe, as well as students who participate in the Google Summer of Code.

A book, *Natural Language Processing with Python*\[^{4}\](Steven Bird and Loper 2009), has been published by O’Reilly Media with the NLTK as its main focus, describing common tasks such as word tagging, text classification, information extraction and syntactic analysis. It is also currently available under a Creative Commons license from the official NLTK website.


\(^3\)Official website, last verified 2nd May 2009: [http://www.nltk.org](http://www.nltk.org)

3.2 MWDumper

MWDumper\(^5\) is a Java tool for processing MediaWiki database dumps. It accepts a MediaWiki XML database dump as input and supports a number of useful filters such as filtering based on MediaWiki namespace, latest pages only and excluding talk pages, before outputting SQL or XML.

3.3 Evan Jones’ wikipedia2text Libraries

In a blog-post\(^6\) on his website, Evan Jones provides a guide on extracting plain-text from Wikipedia articles, and a few libraries to facilitate this process. The modification and use of these tools is further discussed in section 4.2.

3.4 Statistical Tools

Openoffice.org\(^7\) Calc was used for primary inspection spreadsheets produced in the course of the project. For more complex statistical and graphing jobs, SPSS\(^8\) was used. All analytic Python scripts written over the course of the project outputted their data in comma-delimited CSV spreadsheets as both Openoffice.org Calc and SPSS support CSV files natively.

\(^6\)“Extracting Text from Wikipedia”: http://evanjones.ca/software/wikipedia2text.html (Last verified 2nd May 2009)
\(^7\)http://openoffice.org
\(^8\)http://spss.com
Chapter 4

Pre-processing of the corpus

Before actual linguistic analysis of the corpus could begin, a great amount of preparatory work was required. This chapter describes the processes undertaken and methods employed to split the Wikipedia corpus into subcorpora, organized in an orderly directory structure, extract a plaintext version of, and compile s revision history for, each article.

4.1 Compiling Subcorpora

As discussed in section 2.2, the entire Wikipedia corpus comprises too much data to perform any sort of analysis over quickly. It was therefore decided to extract three subcorpora from the entire Wikipedia corpus and perform analysis on them instead.
CHAPTER 4.  PRE-PROCESSING OF THE CORPUS

4.1.1 Criteria for article selection

This plan raised questions almost immediately, however – should the corpora be groups of articles extracted randomly from all over Wikipedia, or should they be coherently grouped sets of articles? If they are grouped, then what should define their grouping?

We decided to create three subcorpora of articles extracted on a non-random basis as this would lead to more coherent, sensible subcorpora being attained. These subcorpora were:

- **“Best” Articles**: A subcorpus such as that described in Evan Jones’ blog-post, this is a collection of articles deemed fit to be included in an “offline release version” of Wikipedia. Around 2722 articles were selected by a Wikipedia group dedicated to building this encyclopedia on the basis of their quality and noteworthiness for version 0.5 of this offline distribution of Wikipedia. The titles of these pages were extracted from the Wikipedia website\(^1\) using a script written by Evan Jones, and then MW-Dumper (see 3.2) was used to filter the entire corpus for these articles.\(^2\)

In the interest of clarity, this corpus is henceforth referred to as the **BA-corpus**.

- **Articles lacking sources**: Articles on wikipedia are organised into Categories as all-encompassing as *1962 Births*, comprised of people born in that year, or even as specific as *Drug-related suicides in California*. There are


\(^2\)Using the command:

```
.bzcat enwiki-20081008-pages-articles.xml.bz2 | java -server -jar mwdumper.jar
\ --format=xml --filter=exactlist:top.txt --filter=latest --filter=notalk
\ > bestarticles.xml
```

where *top.txt* is a list of the articles to include.
also categories dedicated simply to Wikipedia’s up-keep, such as *Articles lacking sources*, a series of categories, chronologically named, filled with articles which have been marked as requiring more citations and sources. A tool (*Category Parser* – see section A.1.1) was scripted to parse the HTML of Category pages and output a list of article names. A series of *Articles lacking sources from Dec 2006* category pages were saved and parsed, and using MWDumper another subcorpus of 2200 articles was compiled.

Henceforth this corpus is referred to as the **LS-corpus**.

- **Articles requiring style editing**: The process performed in order to compile the LS-corpus was repeated, this time using categories full of pages which have been marked as requiring stylistic clean-up. Wikipedia maintains a rather comprehensive manual of style\(^3\) and these pages are typically pages which break a number of these rules and are requiring a considerable re-write. This corpus is comprised of 2356 articles.

Henceforth this corpus is referred to as the **SE-corpus**.

### 4.2 Segmentation of corpus and removal of wiki-markup

Wikipedia pages are written in *wiki mark-up*, a kind of meta-language that is compiled into HTML by the Wikimedia server when a page is requested. This code is used to format text, split articles into sections, insert images and other media and create links to other Wikipedia articles. A sample of it is shown in

Code Listing 4.1

```
"'Frank Vincent Zappa'"<ref name="Francis">Until discovering his birth certificate as an adult, Zappa believed he had been christened "Francis", and he is credited as Francis on some of his early albums. His real name was "Frank", however, never "Francis." Cf. Zappa with Occhiogrosso, 1989, "'The Real Frank Zappa Book"', p. 15.</ref> (December 21, 1940 – December 4, 1993) was an American composer, electric guitarist, record producer and film director. In a career spanning more than 30 years, Zappa wrote rock music, jazz, electronic music, orchestral, and musique concrète works. He also directed feature-length films and music videos, and designed album covers. Zappa produced almost all of the more than 60 albums he released with the band Mothers of Invention and as a solo artist.
```

Code Listing 4.1: Example wiki mark-up

Frank Vincent Zappa\(^{[4]}\) (December 21, 1940 – December 4, 1993) was an American composer, electric guitarist, record producer and film director. In a career spanning more than 30 years, Zappa wrote rock, jazz, electronic, orchestral, and musique concrète works. He also directed feature-length films and music videos, and designed album covers. Zappa produced almost all of the more than 60 albums he released with the band Mothers of Invention and as a solo artist.

Figure 4.1: Rendered wiki-markup.

This presents a problem, since there is, in fact, no full specification for the mark-up language. Thankfully, a set of tools has been written by MIT Ph.D student Evan Jones, which uses the Mediawiki parser (written in PHP) in conjunction with a modified version of BeautifulSoup, wikisoup, to first turn the wiki mark-up into pseudo-XML, and then extract the plaintext. This toolset was heavily modified in places for the purposes of this project. The sources can be reviewed in section A.1.2.
In essence, the XML dump parser takes a single-file XML dump of Wikipedia articles and saves each article page, stripped of its wiki-markup, in its own individual text file in a directory structure. Each directory is named using hashes of the article title, an example being: /bestarticles/4f/31/T-34.txt

All wiki-markup is removed: links are converted to their plain-text equivalents, and all lists and tables are removed. A copy of the original “wikified” article is saved in a text file of the same name, with .markup inserted before the filename extension (e.g. /bestarticles/4f/31/T-34.markup.txt) for the purposes of counting the number of links, images and other Wikipedia-specific characteristics per article.

Since there are some articles which consist of merely one or two introductory sentences followed by long lists (e.g. List of valleys of Malta\(^4\)), the wikisoup library was modified such that it included text within wiki lists. However, the adverse knock-on effects of this were too great\(^5\) and we opted instead to remove these articles from the corpus. From the BA-corpus, 19 articles were removed, from LS-corpus 25 and from SE-corpus 22 articles were removed.


\(^5\)With lists included, the entire text of each list would be considered to be one long sentence. In the interest of keeping the statistical data reliable, these articles were removed.
CHAPTER 4. PRE-PROCESSING OF THE CORPUS

The article in this file is surrounded by minimal XML meta-data preserved from the original article, an example of which can be seen in Code Listing 4.2. This was done in the interest of preserving some meta-data which could prove useful once the time for statistical analysis came. Despite it proving unnecessary, it was retained, since the article text could still be very easily extracted using BeautifulSoup.

The wikisoup parser seems to go into an infinite loop on some articles with complex wiki mark-up and so to combat this, a time-limit on the process was imposed. These articles were few in number, however, and the statistics compiler (see section 5) automatically ignores these aborted articles.

4.3 Compiling Histories

At this point, we had three subcorpora of the English Wikipedia in plaintext, and their accompanying “wikified” versions. However, since we used the XML dump of only the most recent article versions, we had no record of each article’s history.

Thankfully, there is a web application, Daniel Kinzler’s Contributors\textsuperscript{6}, which queries the Wikipedia database, providing a summary of the history of an article’s edits.

\textsuperscript{6}Official website, last verified 2nd May 2009: http://toolserver.org/~daniel/WikiSense/Contributors.php

\begin{lstlisting}[language=xml]
<minor/>
<comment>/* History */</comment>
<text xml:space="preserve">
South Korea, officially the Republic of Korea and often referred to
\end{lstlisting}

Code Listing 4.2: Example head of an article .txt file
To gather this information, a script (see section A.2.1) was written to grab history statistics on each article from the web application. These statistics were saved in .hist files alongside the article .txt file. This script requests histories between 2000 (a year before wikipedia was established) and 2008-10-08, the date on which our snapshot of the Wikipedia database was taken, in HTTP-encoded CSV format. Users are grouped together, edits by bots are ignored, and anonymous and minor edits are counted.

<table>
<thead>
<tr>
<th>User</th>
<th>Start Date</th>
<th>End Date</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mange01</td>
<td>2006-11-17</td>
<td>2007-04-30</td>
<td>+22%</td>
</tr>
<tr>
<td>The+Anomie</td>
<td>2003-02-28</td>
<td>2003-02-28</td>
<td>+23%</td>
</tr>
<tr>
<td>Anomie</td>
<td>2003-02-28</td>
<td>2003-02-28</td>
<td>+23%</td>
</tr>
<tr>
<td>Mange01</td>
<td>2007-11-24</td>
<td>2007-11-24</td>
<td>+10%</td>
</tr>
<tr>
<td>Mlewis000</td>
<td>2006-12-27</td>
<td>2006-12-27</td>
<td>+20%</td>
</tr>
<tr>
<td>91.65.27.163</td>
<td>2007-10-28</td>
<td>2007-10-28</td>
<td>+16%</td>
</tr>
</tbody>
</table>

Code Listing 4.3: Example history file: Rician_fading.hist

Working through the first line of the example hist file shown in Code Listing 4.3, in the first column is the number of edits (6), followed by the user name of the editor – in this case Mange01. This field can also hold an IP address, if the edit was performed anonymously. This is then followed by the date and time of the first edit performed by this user, and ends with the date and time of the last edit performed by the user.

After the maximum number of histories\(^7\) has been collected and stored as .hist files alongside the articles, statistical analysis can begin.

\(^7\)The Contributors application sometimes returns empty pages for articles, so this process was repeated a number of times, with empty .hist files being removed after each pass, until no new history files were being collected.
Chapter 5

Compilation of statistics

5.1 Primary Statistics

The main statistics compilation script (See A.2.2) processes the entire segmented corpus and outputs a comma-delimited CSV file of various statistics for each article. It uses the Punkt sentence tokenizer [Kiss and Strunk 2006] and the standard NLTK word tokenizer. The following primary statistics are compiled:

- **Total Edit Count**, $E_t$
- **Total Author Count**, $A_t$
- **Average Number of Edits per Author**, $E_a$
- **Reduced Edit Count**, $E_r$, the number of edits by authors who have made more than one edit to the article in question.
- **Reduced Author Count**, $A_r$, the number of authors who have made more than one edit to the article in question.
\textbf{CHAPTER 5. COMPILATION OF STATISTICS}

- \textbf{Word Count}, $W_t$
- \textbf{Unique Word Count}, $W_u$
- \textbf{Character Count}, $C_t$, A count of all letters and numbers, excluding white-space and punctuation.
- \textbf{Sentence Count}, $S_t$
- \textbf{Average Wordlength}, $L(W)_a$
- \textbf{Average Sentence Length}, $L(W)_a$

$E_r$ and $A_r$ were recorded as it was feared that statistics for an article with a long tail of editors who contributed only one (perhaps rather insignificant) edit would be distorted.

\section*{5.2 Readability Scores}

Readability tests of each article were also performed. In order to calculate the readability metrics, syllable counts needed be performed. The \LaTeX-hyphenator algorithm and hyphenation patterns [Liang 1983] were used to achieve this. The variables calculated were:

- \textbf{Total Syllable Count}, $\zeta$
- \textbf{Monosyllabic Wordcount}, $\theta$
- \textbf{Polysyllabic Wordcount}, $\delta$
- \textbf{Words of more than 6 characters}, $\varphi$

And the readability tests performed were:
• **Flesch Reading Ease**: A reading test which approximates the “reading ease” of a text. Higher scores of this test [Flesch 1948] indicate material that is easier to read; lower numbers mark passages that are more difficult to read. A score of 90-100 indicates the text is easily understandable by an average 11-year-old student, between 60-70 indicates it is easily understandable by 13 to 15-year-old students and a score of between 0 and 30 indicates it is best understood by college graduates. The formula for the test is:

\[
RE = 206.835 - 1.015 \left( \frac{W_t}{S_t} \right) - 84.6 \left( \frac{\zeta}{W_t} \right)
\]

• **Flesch-Kincaid Grade Level**: This test [Kincaid and Chissom 1975] calculates a readability score of a text to a U.S. grade level, making it easier for teachers and parents to judge the readability level of various books and texts. It can also mean the number of years of education generally required to understand this text, relevant when the formula results in a number greater than 12. The grade level is calculated with the following formula:

\[
GL = 0.39 \left( \frac{W_t}{S_t} \right) + 11.8 \left( \frac{\zeta}{W_t} \right) - 15.59
\]

• **Automated Reading Index**: This test [Kincaid and Chissom 1975] calculates an approximate representation of the U.S. grade level needed to comprehend the text. However, unlike the other indices, the ARI, along with the Coleman-Liau, relies on a factor of characters per word, instead of the usual syllables per word. The reading index is calculated with the
following formula:

\[ ARI = 4.71 \left( \frac{C_t}{W_t} \right) + 0.5 \left( \frac{W_t}{S_t} \right) \]

- **Coleman Liau Readability Score**: The Coleman-Liau Index [Coleman and Liau 1975] is another readability test that produces a U.S. grade-level. It is calculated with the formula:

\[ CL = 5.89 \left( \frac{C_t}{W_t} \right) - 0.3 \left( \frac{S_t}{W_t} \right) - 15.8 \]

- **Gunning FOG**: Devised by Robert Gunning in 1952, Gunning FOG is another readability formula to calculate the number of years of education required to comprehend a text. Its formula is:

\[ GFOG = 0.4 \left( \frac{W_t}{S_t} \right) + 100 \left( \frac{\delta}{W_t} \right) \]

- **SMOG**: Simple Measure of Gobbledygook (SMOG) [McLaughlin 1969] is a readability formula for English which calculates the years of education required to comprehend a text. It is calculated with the formula:

\[ SMOG = 1.0430 \left( \sqrt{30 \left( \frac{\delta}{S_t} \right)} \right) + 3.1291 \]

### 5.3 Further Statistics

Each article was also tagged with the standard NLTK maximum entropy part-of-speech tagger. Analyses of the article text excluding words in “closed categories” (determiners, co-ordinating conjunctions, etc) were also recorded. A
tagged copy of the article is stored too, as is a word-frequency “vocab file” (basically a CSV spreadsheet):

```
South/NNP
Korea/NNP
officially/RB
the/DT
Republic/NNP
of/IN
Korea/NNP
and/CC
often/RB
referred/VBD
to/TO
```

Code Listing 5.1: Excerpt from example part-of-speech data, South_Korea.pos

```
dynasty,2
four,5
facilities,1
prefix,1
Olympics,3
controversial,1
consists,1
oldest,2
Until,1
```

Code Listing 5.2: Excerpt from example word frequency distribution file, South_Korea.vocab

These vocab files are later parsed by the word distribution script (see section A.2.3) which recurses through the corpus directory, creating a spreadsheet of word-length distributions across the whole corpus.
The “wikified” versions of each article are also parsed by the main statistics script, and from these files the following information is extracted:

- **Number of links to other articles**

- **Number of “citation needed” notices**: These are notices placed by Wikipedians in articles where someone feels a sentence needs an inline citation.

- **Number of categories the article belongs to**

- **Number of media objects**: The number of videos and sound files embedded in an article.

- **Number of images**

- **Number of Interwiki Translations**: The number of translations of the article into other languages on Wikipedia.

The standard deviation of sentence lengths and word lengths is calculated using the following formula [Oakes 1998]:

\[
\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2}
\]

Where \( \mu \) is the mean of the sample set.

Functionality was also added to the statistics compiler to generate part-of-speech frequency distribution graphs for each article and for the corpus as a whole. This feature was going to be extended into something more useful but was removed due to time constraints. It is still evident in commented-out lines of code in section A.2.2, however.
5.4 Summary

At the end of this whole process, given an article (e.g. *The Sandlot*) in the corpus, there exist the following files:

- **Article content**,  
  lackingsources/0d/22/The_Sandlot.txt

- **Article content with mark-up**,  
  lackingsources/0d/22/The_Sandlot.markup.txt

- **Article history summary**,  
  lackingsources/0d/22/The_Sandlot.hist

- **Article word-frequency list**,  
  lackingsources/0d/22/The_Sandlot.vocab

- **Article content, tagged with POS-information**,  
  lackingsources/0d/22/The_Sandlot.pos

- **Article content, stripped of closed category words and tagged with POS-information**,  
  lackingsources/0d/22/The_Sandlot-opencategories.pos

We also have the spreadsheet outputted by the `statistics.py` script, with all statistics listed in section 5, and the word-length frequency distribution spreadsheet.

The final size of the corpora after the removal of list articles, articles for which no histories could be obtained and articles which were unparseable is shown in table 5.1.
<table>
<thead>
<tr>
<th>Corpus</th>
<th>Articles</th>
<th>$E_t$</th>
<th>$S_t$</th>
<th>$W_t$</th>
<th>$C_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA-Corpus</td>
<td>2526</td>
<td>4,245,659</td>
<td>431,248</td>
<td>10,203,987</td>
<td>50,882,976</td>
</tr>
<tr>
<td>LS-Corpus</td>
<td>2154</td>
<td>83,238</td>
<td>33,048</td>
<td>721,907</td>
<td>3,522,764</td>
</tr>
<tr>
<td>SE-Corpus</td>
<td>2221</td>
<td>217,039</td>
<td>75,813</td>
<td>1,730,501</td>
<td>8,597,181</td>
</tr>
</tbody>
</table>

Table 5.1: Final size of subcorpora
Chapter 6

Randomized Corpus

6.1 Purpose

Kilgarriff [2005] writes “Language users never choose words randomly, and language is essentially non-random”. Based on this truism, it was decided to compile “randomized corpora” based on the BA-, SE- and LS-corpora, to act as a control in our statistical analyses.

These randomized corpora would be made up of exactly the same number of articles as the original, with each article containing exactly the same number of words as the original article. The only difference would be that these words would be chosen at random from the whole corpus. Two methods of doing this were devised and implemented.


6.2 Preparation

First, a script was written to scan a directory structure for .hist files and concatenate the article text associated with each .hist file into a single corpus text file. The script (see section A.3.1) sought out hist files, instead of the actual article files themselves, since only those articles with histories were being processed for statistics. This script was then executed on each “real” corpus directory, in order to compile concatenated corpora for each of the BA-, SE- and LS-corpora.

6.3 First Method: “Bags Approach”

The first randomizer script does a number of things. It first tokenizes the concatenated “real” corpus by sentence, and to each end-of-sentence marker it assigns the preceding sentence’s length as an extra value. It then deposits all of these annotated end-of-sentence markers in a “bag” and all words in another “bag”. For each article the script finds in the original corpus, it counts the number of complete words in the article and begins picking end-of-sentence markers at random. For each end-of-sentence marker, it grabs at random the number of words associated with it from the bag of words, and creates a new, randomized article. Once a word or an EOS-marker has been taken and placed in the new, randomized article, it is gone from the source bag. See section A.3.2 for a full description of this process, along with the source code.

This approach was first devised due to problems encountered due to the behaviour of the NLTK word tokenizer. These problems were eventually overcome (see section 6.4), but this “randomized-by-bags” corpus was kept, despite not
being absolutely random, in case its statistics yielded anything curious.

The randomized corpora constructed using this method are henceforth referred to as: **RAND-BA-corpus, RAND-LS-corpus** and **RAND-SE-corpus**

### 6.4 Second Method

This randomizer script successfully creates articles, based on those in the real corpus, of randomly chosen words and sentence lengths. Instead of sentence lengths being predetermined by each EOS-particle from the real corpus, words and EOS-particles are thrown in to the same bag and chosen at random. When there are the same number of words in the randomized article as there are in the original, the script proceeds to the next.

The randomized corpora constructed using this method are henceforth referred to as: **RAND2-BA-corpus, RAND2-LS-corpus** and **RAND2-SE-corpus**
Chapter 7

Analysis of results

It is worth mentioning at this point that all data (graphs, spreadsheets, lists, source code) is available on the web at: http://ventolin.org/index.php/academic/

Upon early inspection of the output of the statistics scripts, some interesting statistics of **averages per article** were immediately available for each subcorpus. They are listed in tables 7.1, 7.2 and 7.3.
### Table 7.1: Average values per article per subcorpus, Part 1: Primary Statistics

<table>
<thead>
<tr>
<th>Corpus</th>
<th>$E_t$</th>
<th>$A_t$</th>
<th>$E_a$</th>
<th>$W_t$</th>
<th>$W_a$</th>
<th>$C_t$</th>
<th>$L(W)_a$</th>
<th>$S_t$</th>
<th>$L(S)_a$</th>
<th>$\sigma_{L(S)}$</th>
<th>$\sigma_{L(W)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA-Corpus</td>
<td>1681.45</td>
<td>794.29</td>
<td>2.19</td>
<td>4041.18</td>
<td>1322.81</td>
<td>4.98</td>
<td>170.79</td>
<td>23.71</td>
<td>13.95</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>LS-Corpus</td>
<td>38.66</td>
<td>21.32</td>
<td>1.62</td>
<td>335.3</td>
<td>167.1</td>
<td>4.84</td>
<td>15.35</td>
<td>20.65</td>
<td>10.46</td>
<td>2.76</td>
<td></td>
</tr>
<tr>
<td>SE-Corpus</td>
<td>97.77</td>
<td>46.39</td>
<td>2.59</td>
<td>779.5</td>
<td>344.41</td>
<td>5.04</td>
<td>34.15</td>
<td>22.8</td>
<td>13.24</td>
<td>2.86</td>
<td></td>
</tr>
<tr>
<td>RAND-BA-Corpus</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>4044.48</td>
<td>1973.04</td>
<td>4.94</td>
<td>168.31</td>
<td>23.87</td>
<td>14.57</td>
<td>2.83</td>
<td></td>
</tr>
<tr>
<td>RAND-LS-Corpus</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>335.37</td>
<td>225.04</td>
<td>4.84</td>
<td>15.35</td>
<td>24.01</td>
<td>17.02</td>
<td>2.75</td>
<td></td>
</tr>
<tr>
<td>RAND-SE-Corpus</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>779.63</td>
<td>3834.35</td>
<td>4.92</td>
<td>35.85</td>
<td>20.63</td>
<td>14.12</td>
<td>2.89</td>
<td></td>
</tr>
<tr>
<td>RAND2-BA-Corpus</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>4044.48</td>
<td>1973.04</td>
<td>4.94</td>
<td>168.31</td>
<td>23.87</td>
<td>14.57</td>
<td>2.83</td>
<td></td>
</tr>
<tr>
<td>RAND2-LS-Corpus</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>335.37</td>
<td>224.88</td>
<td>4.83</td>
<td>13.57</td>
<td>24.01</td>
<td>17.02</td>
<td>2.75</td>
<td></td>
</tr>
<tr>
<td>RAND2-SE-Corpus</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>779.63</td>
<td>3834.33</td>
<td>4.93</td>
<td>29.1</td>
<td>27.28</td>
<td>22.74</td>
<td>2.89</td>
<td></td>
</tr>
</tbody>
</table>

### Table 7.2: Average values per article per subcorpus, Part 2: Readability Score Data

<table>
<thead>
<tr>
<th>Corpus</th>
<th>$\zeta$</th>
<th>$\theta$</th>
<th>$\delta$</th>
<th>$\varphi$</th>
<th>$RE$</th>
<th>$GL$</th>
<th>$ARI$</th>
<th>$CL$</th>
<th>$GFOG$</th>
<th>$SMOG$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA-Corpus</td>
<td>6273.07</td>
<td>2640.17</td>
<td>592.35</td>
<td>1133.06</td>
<td>51.54</td>
<td>11.96</td>
<td>13.86</td>
<td>13.5</td>
<td>15.33</td>
<td>13.67</td>
</tr>
<tr>
<td>LS-Corpus</td>
<td>508.06</td>
<td>224.18</td>
<td>44.45</td>
<td>87.95</td>
<td>57.36</td>
<td>10.39</td>
<td>11.67</td>
<td>12.66</td>
<td>13.58</td>
<td>12.17</td>
</tr>
<tr>
<td>SE-Corpus</td>
<td>1200.91</td>
<td>513.21</td>
<td>108.99</td>
<td>214.14</td>
<td>51.36</td>
<td>11.76</td>
<td>13.71</td>
<td>13.88</td>
<td>15.03</td>
<td>13.37</td>
</tr>
<tr>
<td>RAND-BA-Corpus</td>
<td>6250.49</td>
<td>2659.71</td>
<td>585.47</td>
<td>1119.85</td>
<td>51.84</td>
<td>11.96</td>
<td>13.75</td>
<td>13.26</td>
<td>15.34</td>
<td>13.73</td>
</tr>
<tr>
<td>RAND-LS-Corpus</td>
<td>506.53</td>
<td>225.26</td>
<td>44.04</td>
<td>87.14</td>
<td>62.56</td>
<td>8.6</td>
<td>9.54</td>
<td>12.67</td>
<td>11.8</td>
<td>11.27</td>
</tr>
<tr>
<td>RAND-SE-Corpus</td>
<td>1195.84</td>
<td>516.6</td>
<td>107.65</td>
<td>211.5</td>
<td>56.11</td>
<td>10.56</td>
<td>12.04</td>
<td>13.14</td>
<td>13.77</td>
<td>12.66</td>
</tr>
<tr>
<td>RAND2-BA-Corpus</td>
<td>6250.49</td>
<td>2659.71</td>
<td>585.47</td>
<td>1119.85</td>
<td>51.84</td>
<td>11.96</td>
<td>13.75</td>
<td>13.26</td>
<td>15.34</td>
<td>13.73</td>
</tr>
<tr>
<td>RAND2-LS-Corpus</td>
<td>506.52</td>
<td>225.27</td>
<td>44.05</td>
<td>87.13</td>
<td>54.93</td>
<td>11.56</td>
<td>13.32</td>
<td>12.62</td>
<td>14.83</td>
<td>12.91</td>
</tr>
<tr>
<td>RAND2-SE-Corpus</td>
<td>1195.85</td>
<td>516.62</td>
<td>107.66</td>
<td>211.5</td>
<td>49.21</td>
<td>13.17</td>
<td>15.41</td>
<td>13.2</td>
<td>16.44</td>
<td>14.05</td>
</tr>
</tbody>
</table>
Table 7.3: Average values per article per subcorpus, Part 3: Wikipedia-specific Data (Randomized corpora omitted as this data is not applicable to them)

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Links</th>
<th>Citations Needed</th>
<th>Categories</th>
<th>Media</th>
<th>Images</th>
<th>Translations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA-Corpus</td>
<td>290.37</td>
<td>1.73</td>
<td>7.42</td>
<td>0</td>
<td>10.2</td>
<td>50.39</td>
</tr>
<tr>
<td>LS-Corpus</td>
<td>24.95</td>
<td>0.21</td>
<td>2.6</td>
<td>0</td>
<td>0.42</td>
<td>1.63</td>
</tr>
<tr>
<td>SE-Corpus</td>
<td>38.24</td>
<td>0.31</td>
<td>3.22</td>
<td>0</td>
<td>0.82</td>
<td>1.94</td>
</tr>
</tbody>
</table>

7.1 Analysis of preliminary results

There are a few interesting characteristics that are immediately apparent in the tables above. Firstly, it is clear that the amount of work put in to the articles in the BA-corpus far exceeds that put in to articles in the SE- and LS-corpora: the number of edits per article and authors working on each article is far greater, as are the total word count and sentence count.

However, the average number of edits per author in the BA-corpus is 0.4 edits less than the SE-corpus. This is most likely due to the far greater number of authors working on articles in the BA-corpus, perhaps making only one or two edits, with a few key editors making tens or hundreds of edits on a single article.

Another interesting point is the relatively small (but not negligible) lack of variation in $L(S)_a$, the average sentence length, between corpora. This could be evidence of obedience of Wikipedians to the style guide.

The standard deviation of sentence length ($\sigma_{L(S)}$) varies only slightly between corpora, there being a 3.5-word difference between the BA-corpus and LS-corpus. Interestingly, the difference between the two randomized corpora is plain to see: while the randomized corpora built using the “bags approach”
have marginal differences in $\sigma_{L(S)}$ from that of the real corpora, the superiority of the method used to build the RAND2 corpora was plain to see, with the RAND2-SE-Corpus having a $\sigma_{L(S)}$ value 10 words greater than that of the SE-Corpus.

The fact that there is so little difference between the RAND corpora and the real corpora on this point indicates how cohesive the corpora are. A large difference in the value of $\sigma_{L(S)}$ between the two sets of corpora would indicate great variation in sentence length between articles in each of the individual three real corpora.

The readability scores mainly corroborated each other, with articles from the BA- and SE-corpora achieving scores to indicate their suitability for people in their late teens, with the exception being articles in the LS-corpus, which were found to be appropriate for those in their mid-teens.

Perhaps the most interesting result is to be found in the final table: the average number of missing citations per article is over 8 times greater in the BA-corpus than it is in the LS-corpus. This can be put down to the fact that Wikipedians will tend to be far more likely to insert a “citation needed” notice in a polarising article in the BA-corpus (e.g. George W. Bush) than they would in an article in the LS-corpus due to the article’s contentious and popular nature, and the fact that articles in the LS-corpus are already marked with a banner in the article header, notifying readers that the article is in need of citations.

7.2 Word-length frequency distributions

Mendenhall [1887] writes, “preliminary trials in the method [of analysis of mean
word-length] have furnished strong grounds for the belief that it may prove useful as a method of analysis leading to identification or discrimination of authorship”. An analysis of the word-length frequency distribution of each corpus could yield important information in ascertaining how homogenous our corpora are with respect to each other.

Figure 7.1: Word-length frequency distributions for the original corpora

These figures results suggest a high degree of homogeneity between the corpora. The graphs are of the shape typical for a corpus of English text, but exhibit some interesting shared characteristics, such as the “ledge” between the words of 6- and 7-letters, in contrast to the graphs [Mendenhall 1901] of word-length frequencies in Dickens’s Oliver Twist, Thackeray’s Vanity Fair and the works of Shakespeare, Francis Bacon and John Stuart Mill, which all exhibit a slope at this point.

7.3 Sentence-length frequency distributions

Sentence-length distribution curves were also graphed, with an upper bound of 99 words imposed. They are displayed in Fig. 7.2, 7.3 and 7.4.
CHAPTER 7. ANALYSIS OF RESULTS

Figure 7.2: Sentence-length frequencies for the original corpora

(a) BA-corpus
(b) LS-corpus
(c) SE-corpus

Figure 7.3: Sentence-length frequencies for the RAND corpora

(a) RAND-BA-corpus
(b) RAND-LS-corpus
(c) RAND-SE-corpus
CHAPTER 7. ANALYSIS OF RESULTS

(a) RAND2-BA-corpus
(b) RAND2-LS-corpus
(c) RAND2-SE-corpus

Figure 7.4: Sentence-length frequencies for the RAND2 corpora

(a) Jane Austen's *Emma*
(b) Lewis Carroll's *Alice's Adventures in Wonderland*
(c) All Project Gutenberg books packaged with the NLTK (See section A.2.4 for full list)

Figure 7.5: Sentence-length frequencies for some literary works
In the graphs of sentence lengths in the original corpora, we can see just how homogenous the three corpora are in comparison with each other. They all follow the same overall curve, although the curve for the LS-corpus is slightly different, with two peaks occurring at the 15 and 18 word-lengths. If we take the BA and LS-corpora and compare them, articles in the BA-corpus have the highest average number of edits and authors per article. The LS-corpus, on the other hand, has the lowest. While it might seem counterintuitive, these graphs could be seen as evidence that with a greater number of authors and edits, comes a higher level of homogeneity in the corpus. In the BA- and SE-corpora, two corpora whose articles are fairly popular, the peaks occur at the 20-word mark. This is in contrast to the graphs of sentence lengths of literature in Fig. 7.5, whose sentence-length peaks are more towards the 10-12 word mark, compounding the point that popular Wikipedia articles tend towards homogeneity in terms of encyclopedic style.

The RAND set of corpora does not display much randomness due to the way it was compiled. Spikes around the 1 or 2 word mark are caused by the backtracking algorithm in the first corpus randomizer.

The RAND2 corpora, however, are in stark contrast to the original corpora. The curve is almost logarithmic, aside from the dip to the 1-word sentences. It is curious that they are the corpora whose sentence-length frequency curves are most alike the literary sentence-length frequency curves in Fig. 7.5.
7.4 Word Counts

One of the first hypotheses explored was whether or not there is a relationship between the length of an article and the number of people who have worked on it, and the number of revisions it has had. Correlations ($\rho$) between a number of relevant values are shown below in 7.4.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>$\rho(A_t),(W_t)$</th>
<th>$\rho(E_t),(W_t)$</th>
<th>$\rho(A_t),(W_r)$</th>
<th>$\rho(E_r),(W_t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA-Corpus</td>
<td>0.46</td>
<td>0.49</td>
<td>0.46</td>
<td>0.48</td>
</tr>
<tr>
<td>LS-Corpus</td>
<td>0.44</td>
<td>0.51</td>
<td>0.49</td>
<td>0.53</td>
</tr>
<tr>
<td>SE-Corpus</td>
<td>0.44</td>
<td>0.48</td>
<td>0.44</td>
<td>0.47</td>
</tr>
<tr>
<td>All Corpora</td>
<td>0.68</td>
<td>0.67</td>
<td>0.66</td>
<td>0.66</td>
</tr>
</tbody>
</table>

Table 7.4: Correlation between $A_t$ and $W_t$, $E_t$ and $W_t$

These figures demonstrate a correlation of medium significance between the author counts and article length, and edit counts and article length. Across all corpora, there is a correlation of even stronger significance. Ignoring those with only one edit on an article seems to raise this correlation on average by about 0.02, a very marginal difference.

7.5 Correlation between numbers of authors/edits, and lexical richness

One reasonable hypothesis is that with a higher number of contributors to a certain article comes a higher level of lexical variety. As mentioned by Tweedie and Baayen [1998], one way of measuring lexical richness is to measure the type-token ratio, that is, the ratio of unique words ($W_u$) to the total word
count \( (W_t) \). This value was calculated for each article of each corpus, and a correlation between it and the number of authors also calculated.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>( \frac{(\frac{W_a}{W_t})_a}{\rho} )</th>
<th>( \rho[A_t, (\frac{W_a}{W_t})_a] )</th>
<th>( \rho[E_t, (\frac{W_a}{W_t})_a] )</th>
<th>( \rho[A_r, (\frac{W_a}{W_t})_a] )</th>
<th>( \rho[E_r, (\frac{W_a}{W_t})_a] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA-Corpus</td>
<td>0.36</td>
<td>-0.23</td>
<td>-0.24</td>
<td>-0.22</td>
<td>-0.24</td>
</tr>
<tr>
<td>LS-Corpus</td>
<td>0.65</td>
<td>-0.29</td>
<td>-0.31</td>
<td>-0.31</td>
<td>-0.31</td>
</tr>
<tr>
<td>SE-Corpus</td>
<td>0.55</td>
<td>-0.32</td>
<td>-0.32</td>
<td>-0.32</td>
<td>-0.31</td>
</tr>
</tbody>
</table>

Table 7.5: Type-token ratio \( (\frac{W_a}{W_t})_a \); correlation between per-article author count and per-article edit count, and type-token ratio

As seen from the numbers above, the inverse correlations are of small significance and demonstrate, therefore, that there is little direct interaction between this measurement of lexical richness and the number of contributions or contributors to an article. Similar results were produced when authors with only one edit \( (A_r) \) and their edits \( (E_r) \) were ignored.

7.6 Correlation between readability test scores and authors, edit counts

Correlations were also calculated between readability test scores and author and edit counts. Surprisingly, all correlations were of values less than 0.15, signifying lack of any correlation.

The scores were similar when calculated for the reduced author and edit counts \( A_r \) and \( E_r \) and for articles stripped of their closed-category words.
7.7 Standard deviation of sentence length

![Figure 7.6: Sentence-length standard deviations for original corpora](image)

(a) BA-corpus  
(b) LS-corpus  
(c) SE-corpus

Figure 7.6: Sentence-length standard deviations for original corpora

![Figure 7.7: Sentence-length standard deviations for RAND2 corpora](image)

(a) RAND2-BA-corpus  
(b) RAND2-LS-corpus  
(c) RAND2-SE-corpus

Figure 7.7: Sentence-length standard deviations for RAND2 corpora

The stark pattern of the standard deviation in Fig. 7.6b is most striking. While the standard deviation curves for the other corpora are reasonably evenly distributed, the LS-corpus’s forms a perfect diagonal between 20- and 60-words.

7.8 Wikipedia-specific Data

One study that did turn up some strikingly positive results was looking into the correlation between authors, edits and the number of links per article, the
number of translations of an article and the number of “citation needed” notices
in an article.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>$\rho_{\text{LNK}),(A_t)$</th>
<th>$\rho_{\text{LNK}),(E_t)$</th>
<th>$\rho_{\text{CIT}),(A_t)$</th>
<th>$\rho_{\text{CIT}),(E_t)$</th>
<th>$\rho_{\text{TRN}),(A_t)$</th>
<th>$\rho_{\text{TRN}),(E_t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA-Corpus</td>
<td>0.55</td>
<td>0.57</td>
<td>0.12</td>
<td>0.1</td>
<td>0.58</td>
<td>0.53</td>
</tr>
<tr>
<td>LS-Corpus</td>
<td>0.25</td>
<td>0.3</td>
<td>0.11</td>
<td>0.14</td>
<td>0.45</td>
<td>0.4</td>
</tr>
<tr>
<td>SE-Corpus</td>
<td>0.54</td>
<td>0.6</td>
<td>0.19</td>
<td>0.3</td>
<td>0.64</td>
<td>0.58</td>
</tr>
<tr>
<td>All Corpora</td>
<td>0.72</td>
<td>0.73</td>
<td>0.23</td>
<td>0.22</td>
<td>0.75</td>
<td>0.71</td>
</tr>
</tbody>
</table>

Table 7.6: Correlations between $A_t$, $E_t$ and the number of links and missing citation notices on an article and its number of translations

As can be seen in table 7.6, where $\text{LNK}$ is the number of links, $\text{CIT}$ the number of citations and $\text{TRN}$ the number of translations of an article, there are strong correlations between the number of links in an article and the article’s author and edit count.
Chapter 8

Conclusion

This paper has described the process by which plain-text can be extracted from Wikipedia and some interesting statistical analyses that can be carried out on the corpus.

Having taken three fairly representative subcorpora of the entire English Wikipedia corpus, we have shown there is good reason to believe there is a fairly high level of homogeneity, in terms of word-length and sentence-length frequencies, across the corpora. Indeed, homogeneity seems to visibly increase with the amount of work done on an article.

It is surprising, however, to see many of the variables analysed (the readability scores and lexical richness for example) not displaying a direct correlation to author and edit counts. This merits further study, in order to determine what other influences, if any, are coming to bear on these relationships.
8.1 Future work

Unfortunately, due to understandable technical constraints, we were unable to access the entire Wikipedia database as it is. Revisiting the work done in this project with access to the entire database would be worthwhile for a number of reasons.

Firstly, there is no distinction made in the project between major and minor edits, nor is there any distinction made between edits where a contributor has actually contributed something, as opposed to merely removing text. Other edits which don’t actually count as true “contributions” – for example, Wikipedians fixing vandalism, or engaging in an “edit war”, where groups of Wikipedians with opposing views on history revert changes tens of times – could also be easily isolated as such, by analysis of meta-data, such as edit summaries or the timing of edits. Access to the entire database would also remove the dependency on web-applications such as Contributors, the use of which proved to be very time-consuming.

Aside from this, it is my belief that until there is a flexible, stable and fast wiki mark-up lexer, Wikipedia’s valuable, unique content will prove to be an unattractive resource for computational linguists and corpus linguists. There have been some attempts at building alternative lexers and parsers, but these have not yet reached fruition and most have been abandoned.

This is a great pity, since there are many interesting questions left to be answered, such as those relating to the linguistic theory of priming: If a wikipedian, Sal, is in frequent contact with another wikipedian, Pedro, do Pedro’s edits show an influence of Sal’s linguistic traits, and vice-versa?

Another point of interest would be examining the results from section 7.8 in
more detail. Is there a certain “tipping point” at which articles become more “wikified” and laden in wiki mark-up? Indeed, what about the link semantics in Wikipedia? Do the majority of links serve an explanatory or an exploratory purpose? That is to say, do the links encourage users to use the encyclopedia as such, or as a dictionary?

Wikipedia presents a genuinely new and exciting opportunity for corpus linguists, unambiguously free from the constraints of copyright. Once its potential is unlocked by a good toolkit for parsing, it is sure to become a heavily used resource by corpus linguists.
Bibliography


Appendix A

List of Programs

All source code listed in this chapter is available at:

http://ventolin.org/index.php/academic/

A.1 Pre-processing

A.1.1 Subcorpus Filterers

Top Articles Filtering

In order to compile a list of the “top” articles which went on to comprise the BA-corpus, the following bash commands were used to first retrieve webpages listing the articles:

```bash
for i in `seq 1 7` do
done
```
Code Listing A.1: Top Articles Filtering Example

With these pages saved locally (in a directory such as toparticles, as in the following example), the titles of the articles listed were extracted using Evan Jones’ extracttop.py:

```bash
./extracttop.py toparticles/* | sort > top.txt
```

Code Listing A.2: extracttop.py Example Usage

Category Parser

This tool fetches and parses a Wikipedia Category page, outputting a list of all articles found. It is important to note that it does not recurse through the entire Category, but simply parses the one address which it is passed at the command line.

```bash
categoryParser.py http://en.wikipedia.org/wiki/Category:
Articles_lacking_sources_from_July_2008 >> list.txt
```

Code Listing A.3: Category Parser Example Usage

```bash
#!/usr/bin/python
#
# Script to fetch and convert a wikipedia Category page into a list
# of article titles. May not work with all Category pages
# and may break in the future.
#
##
# Example usage:
# categoryParser.py http://en.wikipedia.org/wiki/Category:
# Articles_lacking_sources_from_July_2008 >> list.txt
```
import sys
from BeautifulSoup import BeautifulSoup
import socket
import urllib
import mechanize
from mechanize import UserAgentBase
from mechanize import Browser
import mechanize

def getPage(url):
    cookies = mechanize.CookieJar()
    opener = mechanize.build_opener(mechanize.HTTPCookieProcessor(cookies))
    opener.addheaders = [("User-agent", " Mozilla/5.0 (compatible;.categoryParser.py/0.1)"),("From", "waltona@tcd.ie")]
    mechanize.install_opener(opener)
    r = mechanize.urlopen(url)
    return r.read()

def main():
    page = getPage(sys.argv[1])
    relevance = False
    chunks = ""
    currentLine = ""
    totalLines = []
    for c in page:
        if c == "\n":
            totalLines.append(currentLine)
            currentLine = ""
        else:
            currentLine += c

    for line in totalLines:
        if line.find("table width=\"100\"\") != -1:
            relevance = True
        if line.find("/table") != -1:
Code Listing A.4: Category Parser source code, categoryParser.py

A.1.2 XML Dump Parser

`parseXMLDump.py` is basically a heavily-edited, streamlined version of Evan Jones’ `xmldump2files.py`, `wiki2xml_all.sh` and `wikiextract.py`. It takes a Wikipedia XML dump filename as its first argument, and a directory path as its second argument. It parses the XML database, and for each article it finds, converts the article’s wiki-markup to pseudo-XML using `wiki2xml_command.php` before extracting the plaintext of the article from this pseudo-XML using the `wikisoup` library. It saves each article in a directory structure comprised of two-digit hashes.

If a `-c` switch is passed from the command-line, it skips all articles until an article that hasn’t been processed yet is found. Since this whole process takes
rather a long time (9 hours for the BA-corpus on a 2.4GHz Intel), this can prove useful.

The process depends on finding the wikisoup library in the same directory, and the wiki2xml_command.php file in ./wiki2xml/php/.

Also, it is worth noting that the original wiki2xml_command.php script from Evan Jones’ can sometimes go in to an infinite loop. To combat this, the line set_time_limit(90); was inserted at the top of the file, maximising the lifetime of the process to 90 seconds per article.

```
import md5

#!/usr/bin/python
# Split a Wikipedia XML dump into individual files. The files are stored in a directory tree based on hashing the title of the article.
#
# Remove all wiki markup from article bodies.
# Ignore Talk and User pages
# Output as re-built XML, retaining all wikipedia XML data
#
# Heavily modified by:
# Aengus Walton <waltona@tcd.ie>
# December, 2008
# ventolin.org/index.php/thesis/
#
# Original by:
# Evan Jones <evanj@mit.edu>
# April, 2008
# Released under a BSD licence.
# http://evanjones.ca/software/wikipedia2text.html
```

Code Listing A.5: XML Dump Parser Example Usage
import os
import sys
import urllib
import xml.sax
import wikisoup
import subprocess
import re
import traceback

# Only bother processing/writing if this is set to false:
# (This can probably be done better with MWDumper, but no harm leaving it in)
userOrTalkPage = False

# Are we continuing from a previous job?
continueProcess = False

# Exit code for the faux xml creator
# 0 == success
# 139 == seg fault
# 137 == killed
XMLreturnCode = 0

articleCount=0

randomTMPName = "~/tmp/wikiProcessor" + str(os.getpid())

def clearTmpFiles():
    ""
    Create and initialise files needed by the php script
    ""
    global randomTMPName

    tmp = open(randomTMPName + ".out", "w")
    tmp.write(""
    tmp.close()
    tmp = open(randomTMPName, "w")
    tmp.write(""
    tmp.close()
def writeArticle(self, endReached):
    ""
    Function pseudocode:
    If saving a new article:
        Writes header XML to the new article file in the segmented corpus.
        Converts the wiki-markup'd article to faux-XML.
        Extracts the article plaintext using wiki2xml command.php.
        Saves this to the new article file in the segmented corpus.
    If finishing a previously saved article:
        Append "</revision></page>" to the article.
        Finished.
    ""

global randomTMPName
global continueProcess
global XMLreturnCode
global articleCount

newFile = False
clearTmpFiles()

# ~2.4 million articles at the moment
# assuming an even distribution, we want 2 levels of 2 character directories:
# 3 million / 256 / 256 = 46
# Thus we won't have too many items in any directory

title = self.title.encode("UTF-8")
hash = md5.new(title).hexdigest()
level1 = os.path.join(self.root, hash[0:2])
level2 = os.path.join(level1, hash[2:4])

# Wikipedia-ize the title for the file name

title = title.replace(" ", ",")
title = urllib.quote(title)

# Special case for /: "%x" % ord("/") == 2f
if len(title) > 100:
# fixes an error which can occur with extremely long filenames
# THIS MEANS THE FILENAME CAN’T BE TRUSTED TO BE THE ARTICLE TITLE

title = title [:50]

title += ".txt"
filename = os.path.join(level2, title)

if endReached == False and os.path.exists(filename) == True:
    continueProcess = True
    if articleCount % 100 == 0:
        print "Skipping article #: " + str(articleCount) + ": " + title
    elif continueProcess == True and os.path.exists(filename) == False:
        continueProcess = False

if title[:11] == "Template%3A" or title[:8] == "Image%3A" or title[:12] == "Wikipedia%3A":
    # exclude stuff we don’t want
    continueProcess = True
    if articleCount % 100 == 0:
        print "Skipping article #: " + str(articleCount) + ": " + title

if continueProcess == False:
    if endReached == True:
        out = open(filename, "a")
        out.write("<\revision><\page>"
        out.close
    else:
        if not os.path.exists(level1):
            os.mkdir(level1)
        if not os.path.exists(level2):
            os.mkdir(level2)
        if not os.path.exists(filename):
            newFile = True
        out = open(filename, "a")
        if newFile == True:
            out.write("<\page>\n")
        out.write("<\title>")
out.write(self.title.encode("UTF-8"))
out.write("</title>\n")
out.write("<id>\n")
out.write(self.pageID)
out.write("</id>\n")
out.write("<revision>\n")
out.write("<id>\n")
out.write(self.revID)
out.write("</id>\n")
out.write("<timestamp>\n")
out.write(self.timestamp)
out.write("</timestamp>\n")
out.write("<contributor>\n")
if self.username == None:
    out.write("<ip>\n")
    out.write(self.ip)
    out.write("</ip>\n")
else:
    out.write("<username>\n")
    out.write(self.username.encode("UTF-8"))
    out.write("</username>\n")
    out.write("<id>\n")
    out.write(self.userID)
    out.write("</id>\n")
out.write("</contributor>\n")
if self.minor == True:
    out.write("<minor/>\n")
out.write("</contributor>")
if not self.comment == None:
    out.write(self.comment.encode("UTF-8"))
out.write("</comment>\n")
out.write("<text xml:space="preserve">\n")
tempFileOut = open(randomTMPName, "w")
tempFileOut.write(self.text.encode("UTF-8"))
tempFileOut.close()

if articleCount % 10 == 0:
    print "Processing article #" + str(articleCount) + " - [" + self.title + "]"
APPENDIX A. LIST OF PROGRAMS

### Linux

```python
# XMLreturnCode = subprocess.call(['./createFauxXML.sh', randomTMPName, randomTMPName + '.out'], shell=False)
```

### Windows

```python
nullFile = open('/tmp/null','w')
XMLreturnCode = subprocess.call(['php', 'wiki2xml_command.php', randomTMPName, randomTMPName + '.out'], shell=False, stdout=nullFile, stderr=nullFile)
nullFile.close()
```

### Not Needed When Dealing With Just Current Articles:

```python
if XMLreturnCode != 0:
    tmpfilename = os.path.join(sys.argv[len(sys.argv)]-1, "FAILED.TXT")
    if not os.path.exists(tmpfilename):
        tmpout = open(tmpfilename, "w")
        tmpout.write(str(XMLreturnCode) + "
"
    else:
        tmpout = open(tmpfilename, "a")
        tmpout.write(str(XMLreturnCode) + "
"
    tmpout.close
```

### Try

```python
    try:
        out.write(wikisoup.extractWikipediaText(randomTMPName + ".out").encode("UTF-8"))
        out.write("</text></revision><revision>")
        out.close()
        writeCompleteArticle(filename, self.text.encode("UTF-8"))
    except:
        tmpfilename = os.path.join(sys.argv[len(sys.argv)]-1, "FAILED_WIKISOUP.TXT")
        if not os.path.exists(tmpfilename):
            tmpout = open(tmpfilename, "w")
```

### Exception

```python
```
def writeCompleteArticle(filename, text):
    ""
    Writes the article to /path/articlename.markup.txt, including all wiki markup,
    so that link counts etc. can be calculated later.
    ""
    filename = filename[:-3] + "markup.txt"
    out = open(filename,"w")
    out.write(text)
    out.close()

class WikiPageSplitter(xml.sax.ContentHandler):
    ""
    This class handles the parsing of the XML in the wikipedia XML dump.
    ""
    def __init__(self, root):
        self.root = root
        self.stack = []
        self.text = None
        self.title = None
        self.pageID = None
        self.revID = None
def startElement(self, name, attributes):
    # print "start", name
    if name == "page":
        # assert self.stack == []
        self.text = None
        self.title = None
        self.pageID = None
        self.revID = None
        self.timestamp = None
        self.username = None
        self.userID = None
        self.minor = False  # Is the edit a minor edit?
        self.comment = None
        self.ip = None
    elif name == "title":
        assert self.stack[-1] == "page"
        self.title = ""
    elif name == "id":
        if self.stack[-1] == "page":
            self.pageID = ""
        elif self.stack[-1] == "revision":
            self.revID = ""
    elif name == "timestamp":
        assert self.stack[-1] == "revision"
        self.timestamp = ""
    elif name == "username":
assert self.stack[-1] == "contributor"
self.username = ""

elif name == "ip":
    assert self.stack[-1] == "contributor"
    self.ip = ""

elif name == "minor":
    assert self.stack[-1] == "revision"
    self.minor = True

elif name == "comment":
    assert self.stack[-1] == "revision"
    self.comment = ""

elif name == "text":
    assert self.stack[-1] == "revision"
    self.text = ""

self.stack.append(name)

def endElement(self, name):
    global userOrTalkPage
    global articleCount
    # print "end", name
    if len(self.stack) > 0 and name == self.stack[-1]:
        del self.stack[-1]

    if userOrTalkPage == False and name == "revision" or name == "page":
        # only write to disk if it's not a user/talkpage
        if name == "revision":
            # We have the complete revision: write it out
            writeArticle(self, False)
        elif name == "page":
            articleCount = articleCount+1
            # We have reached the end of the article, append a </page>
            writeArticle(self, True)
        elif userOrTalkPage == True and name == "page":
            #
userOrTalkPage = False

def characters(self, content):
    assert content is not None and len(content) > 0
    global userOrTalkPage
    if len(self.stack) == 0:
        return

    if userOrTalkPage == False:
        if self.stack[-1] == "title":
            if not re.match("User:", content) and not re.match("Talk:", content):
                self.title += content
            else:
                userOrTalkPage = True
        elif self.stack[-1] == "text":
            assert self.title is not None
            self.text += content
        elif self.stack[-1] == "id":
            if self.stack[-2] == "page":
                self.pageID = content
            elif self.stack[-2] == "revision":
                self.revID = content
            elif self.stack[-2] == "contributor":
                self.userID = content
        elif self.stack[-1] == "timestamp":
            self.timestamp += content
        elif self.stack[-1] == "username":
            self.username += content
        elif self.stack[-1] == "comment":
            self.comment += content
        elif self.stack[-1] == "ip":
            self.ip += content

if len(sys.argv) == 1:
    print "Please give either the --stdin flag for processing text from standard input"
    print "followed by a target directory, or pass a source file and target ->
APPENDIX A. LIST OF PROGRAMS


A.2 Corpus Analysis

A.2.1 Contributors Scraper

This script recurses through a directory, finding Wikipedia articles and querying the Contributors web application for their histories. It then saves these histories alongside the articles.

Due to the fact that the Contributors application sometimes returns empty responses or errors, the script checks for these after each run and deletes them.

The script runs in an infinite loop until interrupted by the user with a ^C key-press.

It gives the server a one second rest after each article, so as to not abuse the
service.

```python
#!/usr/bin/python
import time
import sys
import os
import re
import socket
import urllib
import mechanize
from time import sleep
from mechanize import Browser

currentLine = 1
#
# our database compiled on: 2008-10-08

def getArticle(articleName, startDate, endDate, grouped, hideMinor, hideBots, hideAnons, order, max, format):
    
    Use mechanize to interface with the Contributors application, grabbing a history of a certain article.

    br = Browser()
    br.set_handle_robots(False)
    # br.set_proxies({'http': "proxy.cs.tcd.ie:8080"})
    i = 0
    page = None
    # retry 5 times, or until we get the page:
    while page is None or page == "\n" and i < 5:
        page = br.open("http://toolserver.org/~daniel/WikiSense/Contributors.php?wikilang=en&wikifam=wikipedia.org&page=" + articleName + "&since=" + startDate + "&until=" + endDate + "&grouped=" + grouped + "&hideminor=" + hideMinor + "&hidebots=" + hideBots + "&hideanons=" + hideAnons + "&order=" + order + "&max=" + max + "&format=" + format)
        i = i + 1
        sys.stdout.write(".")

    return page.read()
```
def articleHandler(title):
    
    """
    Check if a .hist exists.
    If not, download the history statistics for the article.
    """
    global currentLine

    if os.path.exists(title + ".hist"):
        currentLine = currentLine + 1
        if currentLine % 10 == 0:
            print "#" + str(currentLine) + ": " + title + ".hist exists. ← Skipping..."
        else:
            print "Article #" + str(currentLine) + ": " + title + "\n"

    socket.setdefaulttimeout(10)
    out = open(title + ".hist", "w")

    # strip path out of 'title' if necessary:
    if title.rfind("/") == -1 and title.rfind("\") != -1:
        slashIndex = title.rfind("\")
    else:
        slashIndex = title.rfind("/"")

    try:
        if slashIndex == -1:
            # not found, carry on normally
            out.write(getArticle(urllib.unquote(title),"1995-01-01","←
                                  2008-10-08","on","off","on","off","--edit_count","100000","←
                                  csv")
        else:
            # only query the real article title
            out.write(getArticle(urllib.unquote(title[slashIndex+1:]),"1995-01-01","←
                                  2008-10-08","on","off","on","off","--edit_count","100000","csv")
    except:
        pass
    out.close
    currentLine = currentLine + 1

    # give the server a rest before the next article:
```python
def processDirectory(directory):
    
    """
    Recurse through a directory, looking for articles to find history statistics for
    """
    dirs = [d for d in os.listdir(directory) if os.path.isdir(os.path.join(directory, d))]
    for d in dirs:
        processDirectory(os.path.join(directory, d))

    files = [f for f in os.listdir(directory) if f.endswith(".txt") and not f.endswith(".markup.txt"))]
    for f in files:
        articleHandler(os.path.join(directory, f[-4]))

def processDirForBadHist(directory):
    
    """
    Recurse through a directory, looking for .hist files
    """
    dirs = [d for d in os.listdir(directory) if os.path.isdir(os.path.join(directory, d))]
    for d in dirs:
        processDirForBadHist(os.path.join(directory, d))

    files = [f for f in os.listdir(directory) if f.endswith(".hist")]
    for f in files:
        checkForBadHist(os.path.join(directory, f))

def checkForBadHist(filename):
    
    """
    Check if .hist file is comprised merely of an error message, or is completely empty.
    If it is either, delete it.
    """
    line = ""
    inFile = open(filename,"r")
```
APPENDIX A. LIST OF PROGRAMS

Code Listing A.7: Contributors Scraper, contribScraper.py

A.2.2 Statistics Generator

The main statistics generator recurses through a directory, finding .hist files (since all Wikipedia articles without history statistics should be ignored) and compiling various statistics on each article, before outputting the compiled statistics to a CSV spreadsheet.

If the spreadsheet file already exists, it appends the statistics.

It can also be passed a single .hist file to parse.

Code Listing A.8: Statistics Generator Example Usage
#!/usr/bin/env python
#
# Compile linguistic data on a wikipedia article or group
# of articles and output as a .csv spreadsheet delimited by commas.
#
# Aengus Walton <waltona@tcd.ie>
# March, 2009
# ventolin.org/index.php/thesis/

from __future__ import division
import nltk, re, pprint, os, sys
from BeautifulSoup import BeautifulSoup
from hyphenate import *
import math
import itertools
import pylab

cOMPLETEPOSInfo = [] # part-of-speech info for complete corpus (for graphing, etc)
cOMPLETEOPENPOSInfo = [] # the same, for only open word categories
CURRENТArticle = 0

###### begin hyphenation definitions
hyphenationPatterns = {
    # Knuţh and Liang’s original hyphenation patterns from classic TeX.
    # In the public domain.
    "\ach4 . adder . af1t . al3t . am5at . an5c . ang4 . am5m . ant4 . an3te . anti5s . ar5s
    . ar4tie . ar4ty . as3c . as1p . as1s . aster5 . atom5 . au1d . av4i . awn4 . ba4g . ba5na
    . bas4e . ber4 . be5ra . be3sm . be5sto . bri2 . but4ti . cam4pe . can5c . capa5b . car5ol
    . ca4t . ced4a . ch4 . chill5i . ci2 . cit5r . co3e . co4r . cor5ner . de4moi . de3o . de3ra
    . de3ri . des4c . dictio5 . do4t . du4c . dumb5 . earth5 . eas3i . eb4 . eer4 . eg2 . el5d
    
    # THIS HAS BEEN CUT SHORT IN THE INTEREST OF BREVITY.
def main():
    """
    Decide whether to process an entire directory recursively, or a single article.
    If it seems neither is the case, advise the user on how to use the script from the command line.
    """

    global completePOSInfo
    if len(os.sys.argv) == 3:
        if os.path.isdir(os.sys.argv[1]):
            processDirectory(os.sys.argv[1])
            #print "All done! Plotting and saving a frequency distribution graph of the corpus POS tags..."
            #freqDistCurve = nltk.FreqDist(tag for (word, tag) in completePOSInfo)
            #plotAndSave(freqDistCurve, os.sys.argv[1] + "CorpusPOSFreqDist.png",
                     True, os.sys.argv[1])
            #openFreqDistCurve = nltk.FreqDist(tag for (word, tag) in completeOpenPOSInfo)
            #plotAndSave(openFreqDistCurve, os.sys.argv[1] + "CorpusOpenPOSFreqDist.png", True, os.sys.argv[1] + " (OPEN CATEGORIES)"

    elif os.path.isfile(os.sys.argv[1]):
        processArticle(os.sys.argv[1])
    else:
        print "Something wrong with one of the arguments you passed me...
            "
        print "Does the file / directory exist?"
    else:
        print "Primary Word / History Statistics Generator"
        print "__________________________"
        print "Please provide a directory to recurse through, or a single hist file to parse, as the first argument to this program."
        print "The second argument to the program should be the output..."
def processArticle(filename):
    """
    Compile statistics about an article and store in various variables.
    Complete by sending these variables to saveResults()
    """
    global currentArticle, completeOpenPOSInfo, completePOSInfo
    article = ""
    currentArticle = currentArticle + 1

    print "Processing [" + str(currentArticle) + "]: " + filename[:−5]
    for line in open(filename[:−5] + ".txt", "r").readlines():
        article += line
    # first check that there's an actual body to the article:
    soup = BeautifulSoup(article)
    if soup.text == None or soup.text.contents[0] == "\n" or soup.text.contents[0][:8] == "REDIRECT" or soup.text.contents[0][:9] == "\nREDIRECT":
        print "No text in article, skipping..."
        print ""
    else:
        """ get history statistics:"
        authors, editCount = parseHistFile(filename, False) # full histories
        authorsCut, editCountCut = parseHistFile(filename, True) # only users ->
        with > 1 edit
        """ get link statistics:"
        linkCount, citationNeeded, categoryCount, mediaCount, imageCount, translationCount = getLinkStats(filename)
        """ get word statistics for the full text:
        sentences, strippedWords, strippedText, POSInfo = parseText(soup.text.contents[0])
        sentenceCount = len(sentences)
        wordCount = len(strippedWords)
        charCount = len(strippedText.replace(" ", ""))
        totalSyllables, monosyllables, polysyllables, longWords = calcSyllables(
            (strippedWords, False)
vocablist = buildVocabList(strippedWords)

# now get statistics for only open categories (verbs, nouns, etc):
openPOS = stripClosedCategories(POSInfo)
openWordCount = len(openPOS)
# for the char count, we need to reassemble the text:
newText = reassembleFromPOS(openPOS)
openCharCount = len(newText) - openWordCount
openTotalSyllables, openMonosyllables, openPolysyllables, openLongWords ← calcSyllables(openPOS, True)
# completePOSInfo += POSInfo
# completeOpenPOSInfo += openPOS

strippedWordStdDev = calcStdDev(strippedWords, False)
sentenceStdDev = calcStdDev(sentences, True)

sentenceLengthList = buildSentenceLengthList(sentences)

saveResults(filename, authors, editCount, authorsCut,
             editCountCut, POSInfo, openPOS, sentenceCount,
             wordCount, charCount, totalSyllables, monosyllables,
             polysyllables, longWords, vocablist, openWordCount,
             openCharCount, openTotalSyllables, openMonosyllables,
             openPolysyllables,
             openLongWords, linkCount, citationNeeded, categoryCount,
             mediaCount, imageCount, translationCount,←
             strippedWordStdDev,
             sentenceStdDev, sentenceLengthList)

def saveResults(filename, authors, editCount, authorsCut,
                 editCountCut, POSInfo, openPOS, sentenceCount,
                 wordCount, charCount, totalSyllables, monosyllables,
                 polysyllables, longWords, vocablist, openWordCount,
                 openCharCount, openTotalSyllables, openMonosyllables,
                 openPolysyllables,
                 openLongWords, linkCount, citationNeeded, categoryCount,
                 mediaCount, imageCount, translationCount,←
                 strippedWordStdDev,
                 sentenceStdDev, sentenceLengthList):
Save the results, properly formatted, to the .csv filenames specified upon script execution.

Also save a part-of-speech distribution graph for the article.

(Graph creation & saving currently commented out)

```
global currentArticle

print "Saving results..."

if not os.path.exists(sys.argv[2]):
    outFile = open(sys.argv[2], "w")

outFile.write("NAME," +
           "EDIT COUNT TOTAL," +
           "AUTHOR TOTAL," +
           "AVERAGE EDITS PER AUTHOR," +
           "WORD COUNT," +
           "UNIQUE WORD COUNT," +
           "CHAR COUNT," +
           "AVERAGE WORD LENGTH," +
           "SENTENCE COUNT," +
           "AVERAGE SENTENCE LENGTH (WORDS)," +
           "TOTAL SYLLABLES," +
           "MONOSYLLABIC WORDS," +
           "POLYSYLLABIC WORDS," +
           "LONG (>6 CHAR) WORDS," +
           "OPEN TOTAL SYLLABLES," +
           "OPEN MONOSYLLABIC WORDS," +
           "OPEN POLYSYLLABIC WORDS," +
           "OPEN LONG (>6 CHAR) WORDS," +
           "FLESCH READING EASE," +
           "FLESCH–KINCAID GRADE LEVEL," +
           "AUTOMATED READING INDEX," +
           "COLEMAN LIAU," +
           "GUNNING FOG," +
           "SMOG," +
           "LAESBARHEDSINDEX," +
           "LINSEAR WRITE," +
           "OPEN FLESCH READING EASE," +
           "OPEN FLESCH–KINCAID GRADE LEVEL," +
           "OPEN AUTOMATED READING INDEX," +
           "OPEN COLEMAN LIAU," +
```
APPENDIX A. LIST OF PROGRAMS

"OPEN GUNNING FOG," +
"OPEN SMOG," +
"OPEN LAESBARHEDSINDEX," +
"OPEN LINSEAR WRITE," +
"AUTHOR COUNT (CUT)," +
"EDIT COUNT (CUT)," +
"# OF LINKS TO ARTICLES," +
"# OF MISSING CITATIONS," +
"MEMBER OF x CATEGORIES," +
"MEDIA COUNT," +
"IMAGE COUNT," +
"# OF INTERWIKI TRANSLATIONS," +
"WORD LENGTH STD DEV," +
"SENTENCE LENGTH STD DEV," +

for i in range (1,100):
    outFile.write(str(i) + "-words,"
outFile.write("\n")
outFile.close()

# strip path out of filename if necessary:
slashIndex = 0
if filename.rfind("\\") != -1:
    slashIndex = filename.rfind("\\")
else:
    # if no slash "]" or "/" is found, then the following evaluates to ←
    -1, which works fine with the save function
    slashIndex = filename.rfind("/"

# save main statistics spreadsheet:
outFile = open(sys.argv[2],"a")
try:
    outFile.write(filename[slashIndex+1:-5] + ","
    + str(editCount) + ","
    + str(authors) + ","
    + str(float(editCount)/float(authors)) + ","
    + str(wordCount) + ","
    + str(len(vocablist)) + ","
    + str(charCount) + ","
    + str(float(charCount)/float(wordCount)) + "\n")
+ str(sentenceCount) + ","
+ str(float(wordCount)/float(sentenceCount)) + ","
+ str(totalSyllables) + ","
+ str(monoSyllables) + ","
+ str(polySyllables) + ","
+ str(longWords) + ","
+ str(openTotalSyllables) + ","
+ str(openMonoSyllables) + ","
+ str(openPolySyllables) + ","
+ str(openLongWords) + ","
+ str(fleschReadingEase(totalSyllables, wordCount, sentenceCount)) + ","
+ str(fleschGradeLvl(totalSyllables, wordCount, sentenceCount)) + ","
+ str(ari(wordCount, sentenceCount, charCount)) + ","
+ str(colemanLiau(wordCount, sentenceCount, charCount)) + ","
+ str(fog(wordCount, sentenceCount, polysyllables)) + ","
+ str(smog(sentenceCount, polysyllables)) + ","
+ str(lix(wordCount, sentenceCount, longWords)) + ","
+ str(linear(wordCount, sentenceCount, polysyllables)) + ","
+ str(fleschReadingEase(openTotalSyllables, openWordCount, sentenceCount)) + ","
+ str(fleschGradeLvl(openTotalSyllables, openWordCount, sentenceCount)) + ","
+ str(ari(openWordCount, sentenceCount, openCharCount)) + ","
+ str(colemanLiau(openWordCount, sentenceCount, openCharCount)) + ","
+ str(fog(openWordCount, sentenceCount, openPolysyllables)) + ","
+ str(smog(sentenceCount, openPolysyllables)) + ","
+ str(lix(openWordCount, sentenceCount, openLongWords)) + ","
+ str(linear(openWordCount, sentenceCount, openPolysyllables)) + ","
+ str(authorsCut) + ","
APPENDIX A. LIST OF PROGRAMS

+ str(editCountCut) + ","
+ str(linkCount) + ","
+ str(citationNeeded) + ","
+ str(categoryCount) + ","
+ str(mediaCount) + ","
+ str(imageCount) + ","
+ str(translationCount) + ","
+ str(strippedWordStdDev) + ","
+ str(sentenceStdDev) + ",")

# sentence length capped at 100 words:
for i in range (1,100):
    try:
        outFile.write(str(sentenceLengthList[i]["counter"]) + ",")
    except KeyError:
        outFile.write("0,")
outFile.write("\n")

except ZeroDivisionError:
    print "There was a divide−by−zero error. Not including this article in the results."
outFile.close()

# save word frequency list:
outFile = open(filename[−5] + "−vocab","w")
for k, v in vocablist.items():
    outFile.write(k.encode("UTF−8") + " ", str(v['count']) + "\n")
outFile.close()

# save part−of−speech analysis:
outFile = open(filename[−5] + "−pos","w")
for k in POSInfo:
    outFile.write(k[0].encode("UTF−8") + "/ " + k[1].encode("UTF−8") + "\n")
outFile.close()
outFile = open(filename[−5] + "−opencategories.pos", "w")
for k in openPOS:
    outFile.write(k[0].encode("UTF−8") + "/ " + k[1].encode("UTF−8") + "\n")
outFile.close()
# save a graph of the part-of-speech distribution for each article:
#freqDistCurve = nltk.FreqDist(tag for (word, tag) in POSInfo)
#plotAndSave(freqDistCurve, filename[-5] + "-POSfreqdist.png", True,"POS ←
Distrib. Curve for: " + filename[slashIndex+1:-5])
#openFreqDistCurve = nltk.FreqDist(tag for (word, tag) in openPOS)
#plotAndSave(openFreqDistCurve, filename[-5] + "-openPOSfreqdist.png", True,"POS Distrib. Curve for: " + filename[slashIndex+1:-5] + " (OPEN←
CATEGORIES)")

print "" # all done, on to the next one..

def getLinkStats(filename):
    ""
    Count, and return in the following order, these statistics:
    
    links: number of links to other wikipedia articles in this article.
citation: number of "[citation needed]" notices in the article.
categories: number of categories this article is a member of.
media: number of media items in the article (sound, video, etc)
images: number of images in the article.
translation: number of interwiki translations of this article.
    ""
    # links: [[text]]
    # category links: [[Category:text]]
    # media links: [[media:text]]
    # images: [[Image:filename]]
    # citation needed: {{Fact|date}}
    # interwiki translation: [[XX:text]] (works for most interwikis with 2–3←
letter language codes)
    
    text = ""
    for line in open(filename[-5] + ".markup.txt","r").readlines():
        text += line
    p = re.compile('\\\[\[Category:.+?\]\]]\}')
categories = len(p.findall(text))
p = re.compile('\\\[\[Image:.+?\]\]]\}')
images = len(p.findall(text))
p = re.compile('\\\[\[media:.+?\]\]]\}')
media = len(p.findall(text))
p = re.compile('\[\[.\?:\:.+?\]\]')
translation = len(p.findall(text))
p = re.compile('\[\[.\+:\:.+\]\]')
links = len(p.findall(text))
links = links - categories - images - media - translation
p = re.compile('{{Fact.+}}')
citation = len(p.findall(text))
return links, citation, categories, media, images, translation

def parseHistFile(filename, cutTail):
    ""
    Parse a .hist file, compiling statistics on the # of edits, # of authors, etc.
    If the cutTail boolean is set to "True", then count these values, but ignore all
    authors who have contributed only one edit.
    ""
    editCount = 0
    authors = 0

    inFile = open(filename,"r")
    try:
        for line in inFile:
            if not cutTail:
                if line.strip():
                    #maj = maj + int(line[line.find("+%28") +4:line.find("%2F")])
                    #min = min + int(line[line.find("%2F") +3:line.find("%29.")])
                    editCount += int(line[:line.find("",)])
                    authors = authors + 1;
                else:
                    pass # line is empty
            else:
                #ignore users who only made 1 edit
                if line.strip():
                    tmp = int(line[:line.find("",)])
                    if tmp > 1:
editCount += tmp
authors = authors + 1
else:
    pass

finally:
inFile.close()
return authors, editCount

def parseText(text):
    ""
    Sententially tokenize the article.
    Strip the article of all punctuation (apart from apostrophes) and tokenize each sentence by words.
    POS-Tag each sentence.
    Return all the data.
    ""
    print "Tokenizing & analysing sentences..."
sentence_tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
sentences = sentence_tokenizer.tokenize(text)
strippedText = removePunctuation(text)
strippedText = strippedText.replace("\n"," ")
print "Tokenizing & analysing words...",strippedWords = nltk.word_tokenize(strippedText)
POSInfo = nltk.pos_tag(strippedWords)

return sentences, strippedWords, strippedText, POSInfo

def removePunctuation(text):
    ""
    Remove all exclamation marks, full stops, question marks, quotation marks, parentheses, colons, semi-colons and commas from a string and return the modified string.
    ""
    result = ""
    for c in text:
        if c in '!.?()::,':
```python
def calcSyllables(words, POS):
    ""
    Calculate the number of syllables in a tokenized text and return various relevant information:
    totalSyllables: total syllable count of the article
    monosyllables: total number of monosyllabic words
    polysyllables: total number of polysyllabic (3 or more syllables) words
    longWords: total number of words longer than 6 characters.
    
    From these numbers, the various readability tests can be carried out.
    
    If the POS boolean is set to be true, this function expects that "words" is not just tokenized text, but tokenized and POS-tagged, and acts accordingly.
    ""
    global hyphenationPatterns
    global hyphenationExceptions
    totalSyllables = 0
    monosyllables = 0
    polysyllables = 0
    longWords = 0
    hyphenator = Hyphenator(hyphenationPatterns, hyphenationExceptions)
    if POS == False:
        for word in words:
            syllableCount = len(hyphenator.hyphenate_word(word))
            totalSyllables += syllableCount
            if syllableCount < 2:
                monosyllables += 1
            elif syllableCount >= 3:
                polysyllables += 1
            if len(word) > 6:
                longWords += 1
        elif POS == True:
            pass
    else:
        result += c
    return result
```
for j, k in words:
    syllableCount = len(hyphenator.hyphenate_word(str(j.encode("UTF-8")))))
    totalSyllables += syllableCount
    if syllableCount < 2:
        monosyllables += 1
    elif syllableCount >= 3:
        polysyllables += 1
    if len(str(j.encode("UTF-8"))) > 6:
        longWords += 1
return totalSyllables, monosyllables, polysyllables, longWords

def buildVocabList(words):
    """
    Parses word-tokenized text and returns a python dictionary of unique vocabulary
    paired with the number of occurrences of each word in the text.
    """
    vocablist = {}
    for w in words:
        if not vocablist.has_key(w):
            vocablist[w] = {}
        try:
            vocablist[w]['count'] = vocablist[w]['count'] + 1
        except KeyError:
            vocablist[w]['count'] = 1
    return vocablist

def buildSentenceLengthList(sentences):
    """
    Parses sentence-tokenized text and returns a python dictionary of sentence lengths paired with the number of occurrences.
    """
    dictionary = {}
    for sentence in sentences:
        length = len(nltk.word_tokenize(removePunctuation(sentence)))
        if not dictionary.has_key(length):
            dictionary[length] = {}
        try:
APPENDIX A. LIST OF Programs

```python
dictionary[length]["counter"] = int(dictionary[length]["counter"]) + 1

except KeyError:
dictionary[length]["counter"] = 1
return dictionary

def reassembleFromPOS(POSInfo):
    """
Parses a POS−tagged text and returns an un−tagged text.
"""
    text = ""
    for w, p in POSInfo:
        text += str(w.encode("UTF−8")) + " "
    return text[:-1] # there'll be an extra space at the end, best chop it off...

def stripClosedCategories(POSInfo):
    """
Parses a POS−tagged text and returns a POS−tagged text of all words belonging to open categories (all that aren't determiners, co−ordinating conjs, prepositions, etc)
"""
    newText = []
    for word in POSInfo:
        if word[1] == "DT" or word[1] == "IN" or word[1] == "CC" or word[1] == "TO" or word[1] == "POS" or word[1] == "WDT" or word[1] == "PRP" or word[1] == "WP" or word[1] == "WRB":
            pass # these are categories we're ignoring
        else:
            tempList = (word[0],word[1])
            newText.append(tempList)
    return newText

def calcStdDev(dataset, areTheySentences):
    """
    Calculate the standard deviation of a dataset (set of sentences, words, etc)
    based on their length (# of words in sentence, # of characters per word, etc)
    """
```
if not areTheySentences:
    # we're dealing with std deviation of wordlengths
    sum = 0
    for item in dataset:
        sum += len(item)
    mean = float(sum/len(dataset))

    squaredDiff = []
    for item in dataset:
        squaredDiff.append(float((len(item) - mean)*(len(item) - mean)))

    sum = 0
    for item in squaredDiff:
        sum += item
    return float(math.sqrt(sum/len(dataset)))
else:
    # we're dealing with std deviation of sentence lengths
    sum = 0
    for item in dataset:
        # don't strip punctuation - since punctuation in the randomized ←
        # corpus is left in.
        sum += len(nltk.word_tokenize(item))
    mean = float(sum/len(dataset))

    squaredDiff = []
    for item in dataset:
        squaredDiff.append(float((len(nltk.word_tokenize(item)) - mean)←
                                 *(len(nltk.word_tokenize(item)) - mean)))

    sum = 0
    for item in squaredDiff:
        sum += item
    return float(math.sqrt(sum/len(dataset)))

def processDirectory(directory):
    """
    Processes a directory recursively, looking for .hist files to parse.
    """
dirs = [d for d in os.listdir(directory) if os.path.isdir(os.path.join(directory, d))]
for d in dirs:
    processDirectory(os.path.join(directory, d))

files = [f for f in os.listdir(directory) if f.endswith(".hist")]
for f in files:
    processArticle(os.path.join(directory, f))

# statistical readability formulae:
def fleschReadingEase(totalSyllables, wordCount, sentenceCount):
    
    """
    Flesch Reading Ease
    """
    return 206.835 - (totalSyllables * 84.6 / wordCount) - (wordCount * 1.015 / sentenceCount)

def ari(wordCount, sentenceCount, charCount):
    
    """
    Automated Reading Index (ARI)
    """
    return (4.71 * charCount / float(wordCount)) + (0.5 * float(wordCount) / float(sentenceCount)) - 21.43

def fleschGradeLvl1(totalSyllables, wordCount, sentenceCount):
    
    """
    Flesch–Kincaid Grade Level
    """
    return (0.39 * wordCount / float(sentenceCount)) + (11.8 * totalSyllables / wordCount) - 15.59

def colemanLiau(wordCount, sentenceCount, charCount):
    
    """
    Coleman Liau
    'A computer readability formula designed for machine scoring', Journal of Applied Psychology. (1975)
    """
    return (5.89 * charCount / float(wordCount)) - (0.3 * float(sentenceCount))
APPENDIX A. LIST OF PROGRAMS

sentenceCount / float(wordCount) - 15.8

def fog(wordCount, sentenceCount, polysyllables):
    """
    Gunning Fog, will change when I find Gunning original
    """
    return 0.4 * ((float(wordCount) / float(sentenceCount)) + 100. * (→
        float(polysyllables) / float(wordCount)) )

def smog(sentenceCount, polysyllables):
    """
    Simple Measure Of Gobbledygook
    """
    return 1.0430 * math.sqrt(polysyllables*30.0/(sentenceCount)) + 3.1291

def lix(wordCount, sentenceCount, longWords):
    """
    Laesbahnshsindex (Any language)
    """
    return (wordCount/float(sentenceCount)) + ((100. * longWords)/float(→
        wordCount))

def linsear(wordCount, sentenceCount, polysyllables):
    """
    Linsear Write
    """
    hardwords = polysyllables
    easywords = wordCount - hardwords
    hardwords += 3
    r = hardwords+easywords
    r /= sentenceCount
    if r > 20:
        r /= 2
    elif r <= 20:
        r -= 2
        r /= 2
    return r

# end of readability formulae

def plotAndSave(data, filename, cumulative, title):
    """
    modified version of the nltk FreqDist.plot() method, enabling saving to a file and disabling
    its dialog box display, to make the whole process automatic.
    """
    samples = list(islice(data, len(data)))

    if cumulative:
        freqs = list(data._cumulative_frequencies(samples))
        ylabel = "Cumulative Counts"
    else:
        freqs = [data[sample] for sample in samples]
        ylabel = "Counts"

    pylab.grid(True, color="silver")
    pylab.plot(freqs, linewidth=2)
    pylab.xticks(range(len(samples)), [str(s) for s in samples], rotation=90)
    pylab.title(title)
    pylab.xlabel("Samples")
    pylab.ylabel(ylabel)
    pylab.savefig(filename)

main()

Code Listing A.9: Statistics Generator, statistics.py

A.2.3 Word-length Frequency Generator

This is a simple tool which iterates through a directory, parsing the vocab files it finds which have been built by the statistics generator, and saves a CSV spreadsheet of word-length frequencies of each article of a corpus.
Code Listing A.10: Word-length Frequency Example Usage

```python
#!/usr/bin/python
# Compile a spreadsheet of the number of 1-, 2-, 3-, 4-letter words (up to 100-letters)
# per wikipedia article, and save as a .csv spreadsheet delimited by commas.
#
# Aengus Walton <waltona@tcd.ie>
# March, 2009

from __future__ import division
import re, pprint, os, sys
from BeautifulSoup import BeautifulSoup

currentArticle = 0

def main():
    if len(sys.argv) < 2:
        printHelp()
    else:
        if os.path.isdir(sys.argv[1]):
            processDirectory(sys.argv[1])
        elif os.path.isfile(sys.argv[1]):
            processArticle(sys.argv[1])
        else:
            printHelp()

def printHelp():
    """
    Print usage information
    """
    print "Word Length Distribution calculator."
    print "--------------------"
    print "Please provide a directory to recurse through, or a single"
    print ".vocab file to parse, as the first argument to this program."
    print "\nThe second argument to the program should be the output file"
```
print " for the main . csv statistics spreadsheet. \n"

def processDirectory(directory):
    """
    Recurse through a directory, finding . vocab files to process.
    """
    dirs = [d for d in os.listdir(directory) if os.path.isdir(os.path.join(directory, d))]
    for d in dirs:
        processDirectory(os.path.join(directory, d))

    files = [f for f in os.listdir(directory) if f.endswith(" . vocab")]
    for f in files:
        processArticle(os.path.join(directory, f))

def processArticle(filename):
    """
    Process the vocab file, creating a word-length frequency list.
    Save as a . csv file
    """
    global currentArticle
    currentArticle = currentArticle + 1
    dictionary = {}

    # for isolating the article title without its full path:
    if filename.rfind(" \") != -1:
        slashIndex = filename.rfind(" \")
    else:
        # if no slash " \" or "/" is found, then the following evaluates to -1, which works fine with the save function
        slashIndex = filename.rfind(/")

    if currentArticle % 10 == 0:
        print " Currently Processing [ # + str(currentArticle) + "]: " + ↔
        filename[slashIndex + 1:]
    for line in open(filename):
        word, count = line.split(" ,")
        wordlength = str(len(word))
        if not dictionary.has_key(wordlength):
```python
dictionary[wordlength] = {}
try:
    dictionary[wordlength]["counter"] = int(dictionary[wordlength] ← ["counter"]) + int(count)
except KeyError:
    dictionary[wordlength]["counter"] = int(count)

if not os.path.exists(sys.argv[2]):
    outFile = open(sys.argv[2],"w")
    # write header of csv file:
    outFile.write("NAME, ")
    for i in range (1,100):
        outFile.write(str(i) + "-letters," )
    outFile.write("\n")
    outFile.close()
outFile = open(sys.argv[2],"a")
outFile.write(filename[slashIndex+1:] + ",")

# max length of word capped at 100 letters.
# this loop ensures counts remained properly aligned in the spreadsheet
for i in range (1,100):
    try:
        outFile.write(str(dictionary[str(i)]["counter"]) + "," )
    except KeyError:
        outFile.write("0, ")
    outFile.write("\n")
outFile.close()

main()
```

Code Listing A.11: Word-length Frequency Generator, worddistrib.py
A.2.4  Project Gutenberg Sentence-length Frequency Generator

This script compiles sentence length statistics for a directory of text files. The directory used was the NLTK’s selection from Project Gutenberg\(^1\). It contained the full texts of the following works:

- *Emma* by Jane Austen (1816)
- *Persuasion* by Jane Austen (1818)
- *Sense and Sensibility* by Jane Austen (1811)
- The Bible (King James Version)
- *Songs of Innocence and of Experience* and *The Book of Thel* by William Blake (1789)
- *Stories to Tell to Children* by Sarah Cone Bryant (1918)
- *The Adventures of Buster Bear* by Thornton W. Burgess (1920)
- *Alice’s Adventures in Wonderland* by Lewis Carroll (1865)
- *The Ball and The Cross* by G.K. Chesterton (1909)
- *The Wisdom of Father Brown* by G. K. Chesterton (1914)
- *The Man Who Was Thursday* by G. K. Chesterton (1908)
- *The Parent’s Assistant* by Maria Edgeworth
- *Moby Dick* by Herman Melville (1851)
- *Paradise Lost* by John Milton (1667)
- *The Tragedie of Julius Caesar* by William Shakespeare (1599)

\(^1\)http://www.projectgutenberg.org
The Tragedie of Hamlet by William Shakespeare (1599)

The Tragedie of Macbeth by William Shakespeare (1603)

Leaves of Grass by Walt Whitman (1855)
print "Tokenizing, building sentence length list for all files"

sentenceLengthList = buildSentenceLengthList(sentence_tokenizer.tokenize(everything))

print "writing csv file..."
outFile = open("nltk_corpora_sentlength.csv","w")
# sentence length capped at 100 words:
for x in range (1,100):
    try:
        outFile.write(str(sentenceLengthList[x]["counter"] + ",")
    except KeyError:
        outFile.write("0,")
    outFile.write("\n")
main()

Code Listing A.12: Project Gutenberg Sentence-length Frequency Generator, gutenberg.sentencelengths.py

A.3 Randomized Corpus Generation

A.3.1 Corpus Concatenator

This script simply recurses through a segmented Wikipedia corpus, looking for history files. For any history file it finds, it appends the associated article to a single file.

The result is a corpus of Wikipedia plain-text in a single file, comprised of only those articles which have been processed by the statistics generator, ready to be randomized by the corpus randomizer scripts (see sections A.3.2 and A.3.3).
#!/usr/bin/env python
#
# Concatenate a corpus of split Wikipedia articles for which there exists a
# .hist file into one file.
#
# Aengus Walton <waltona@tcd.ie>
# April 2009

from BeautifulSoup import BeautifulSoup
import os, sys
fileCount = 0

def processDirectory(directory):
    dirs = [d for d in os.listdir(directory) if os.path.isdir(os.path.join(directory, d))]
    for d in dirs:
        processDirectory(os.path.join(directory, d))

files = [f for f in os.listdir(directory) if f.endswith(".hist")]
for f in files:
    processArticle(os.path.join(directory, f))

def processArticle(filename):
    global fileCount
    filename = filename[:-5] + " .txt"
    fileCount = fileCount + 1
    fileContents = ""
    if fileCount % 10 == 0:
        print "Processing #" + str(fileCount)
    for line in open(filename):
        fileContents += line
    soup = BeautifulSoup(fileContents)
    if soup.text == None or soup.text.contents[0] == "\n" or soup.text.←
Code Listing A.14: Corpus Concatenator, concatenator.py

### A.3.2 First Method

This is an outline of how the first corpus randomizer works:

1. Read in the concatenated corpus and tokenize it by sentences.
2. Create two “bags” (Python lists), one for words and the other for full-stops.
3. For each sentence, count the wordlength and attach it as an attribute to that sentence’s full-stop.
4. Put all annotated EOS-markers in EOSBag and all words in WordBag.
5. Recurse through the corpus and for each article:
   - Count the number of words in the article.
• Pick a random EOS-marker and append to the new, randomized article, the number of words associated with it. Append a full-stop.

• Subtract this number of words from the number of words remaining to be added to the article

• Repeat this process until the article’s word-count has been met. If too many words have been added, place the last sentence’s words back in the WordBag, and the EOS-marker back in the EOSBag, and look for an EOS-marker, at random, with a shorter word-count associated with it.

This program takes 3 arguments: the root directory of the original, real corpus; the output directory of the new, randomized corpus; and the text file containing the concatenated, real corpus.

```
randomizer.py c:\fyp\xml\bestarticles c:\fyp\xml\randomized_best c:\fyp\←
bestarticles.txt
```

Code Listing A.15: Corpus Randomizer Example Usage

```python
#!/usr/bin/env python
# Corpus randomizer
# Create a corpus of the same number of articles as a 'real' corpus, each article having an identical word-count to the original article but ← comprised of random words.
##
# This is the "WordBag" / "EOS-bag" approach
##
# Aengus Walton <waltona@tcd.ie>
# April, 2009
##
```
import os, sys
from BeautifulSoup import BeautifulSoup
import nltk
from random import randint

def processDirectory(directory):
    dirs = [d for d in os.listdir(directory) if os.path.isdir(os.path.join(directory, d))]
    for d in dirs:
        processDirectory(os.path.join(directory, d))

    files = [f for f in os.listdir(directory) if f.endswith(".hist")]
    for f in files:
        articleCounter = articleCounter + 1
        print "Processing: #" + str(articleCounter)
        processArticle(directory, f[:−5] + ".txt")

def processArticle(directory, f):
    ""
    Create the randomized article and save it
    """
    global corpusWords
    global corpusWordLength
    global wordBag
    global EOSBag
    remainingWordCount = getWordCount(os.path.join(directory, f))

    if remainingWordCount == 0:
        print "PASS."
    else:
        # find an appropriate place to put the new article file, similar
        # to that found in the original corpus:
        level1 = str(directory)[len(str(directory))−5:len(str(directory))−3]
        level2 = str(directory)[len(str(directory))−2:]
        if not os.path.exists(os.path.join(sys.argv[2], level1, level2)):
            os.mkdir(os.path.join(sys.argv[2], level1))
        if not os.path.exists(os.path.join(sys.argv[2], level1, level2,level2)):
            os.mkdir(os.path.join(sys.argv[2], level1, level2,level2))
newFilename = os.path.join(sys.argv[2], level1, level2, f)

newArticleList = []
currentEOS = []
delthis = 0

while remainingWordCount != 0:
    if remainingWordCount < 0:
        # return last sentence to wordBag / EOSBag
        lastSentenceLength = newArticleList[len(newArticleList) - 1][1]
        EOSBag.append(newArticleList[len(newArticleList) - 1])
        del newArticleList[len(newArticleList) - 1]
        for i in range(0, lastSentenceLength):
            wordBag.append(newArticleList[len(newArticleList) - 1])
            del newArticleList[len(newArticleList) - 1]
        remainingWordCount += lastSentenceLength

    # look for a shorter sentence than the last:
    foundShorterSentence = False
    while foundShorterSentence == False:
        if len(EOSBag) - 1 == 0:
            x = 0
            print "Something's gone horribly wrong!"
            print "There are no short sentences to finish this article with."
            print "Breaking..."
            break
        else:
            x = randint(0, len(EOSBag) - 1)
            if EOSBag[x][1] < lastSentenceLength:
                currentEOS = EOSBag[x]
                foundShorterSentence = True
            else:
                # randint() doesn't like trying to find a random number between 0 and 0:
                if len(EOSBag) - 1 == 0:
                    x = 0
                else:
x = randint(0, len(EOSBag)−1)
currentEOS = EOSBag[x]
del EOSBag[x]

for i in range(0, currentEOS[1]):
    if len(wordBag)−1 == 0:
        x = 0
    else:
        # python’s own version of the do {...} while loop:
        # we only want random words — don’t want any full stops Cree pin in.
        while True:
            x = randint(0, len(wordBag)−1)
            if not type(wordBag[x]) == list: break
        newArticleList.append(wordBag[x])
        del wordBag[x]
        newArticleList.append(currentEOS)
        remainingWordCount = remainingWordCount − currentEOS[1]
delthis = delthis + 1

newArticle = ""
    # very unpythonic, but this "for i in range" loop prevents sentences like:  
    # "the and Kitchen . new sentence here . full stops wrong aligned no .
    for i in range(0, len(newArticleList)):
        if type(newArticleList[i]) == list:
            newArticle += ". "
        else:
            if type(newArticleList[i+1]) == list:
                newArticle += newArticleList[i]
            else:
                newArticle += newArticleList[i] + ". 

outFile = open(newFilename, "w")
outFile.write(newArticle)
outFile.close()

def getWordCount(filename):
inFile = open(filename, "r")
APPENDIX A. LIST OF PROGRAMS

```python
lines = ""
for line in inFile:
    lines += line
soup = BeautifulSoup(lines)
if soup.text == None or soup.text.contents[0] == "\n" or soup.text.contents[0][0:8] == "REDIRECT" or soup.text.contents[0][0:9] == "\nREDIRECT":
    return 0
else:
    words = nltk.word_tokenize(removePunctuation(soup.text.contents[0]))
    return len(words)

def removePunctuation(text):
    """
    Remove all exclamation marks, full stops, question marks, quotation marks, parentheses, colons, semi-colons and commas from a string and return the modified string.
    """
    result = ""
    for c in text:
        if c in '!'?.()";,':
            pass
        else:
            result += c
    return result

def main():
    global corpusWordLength
    global wordBag
    global EOSBag
    if len(sys.argv) < 3:
        print "Corpus Text Randomizer"
        print """"""
        print "Takes a corpus of texts, split on the basis of individual articles"
        print "and constructs a similar corpus of randomized text."
```
print "Each article will have the same number of words as the original,"
print "but a random number of words per sentence."
print "Usage:
print "randomizer.py [root dir of original corpus] [output dir of randomized corpus] [textfile containing complete original corpus]"
else:
    # tokenize entire corpus .txt file:
    print "Reading in full corpus file and tokenizing sentences..."
    sentence_tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
    corpusText = ""
    for line in open(sys.argv[3]):
        corpusText += line
    print "Tokenizing words..."
    sentences = sentence_tokenizer.tokenize(corpusText)
    print "Creating bags of words and EOS particles..."
    for sentence in sentences:
        words = nltk.word_tokenize(removePunctuation(sentence))
        EOSBag.append([".", len(words)])
        for word in words:
            wordBag.append(word)
    corpusWordLength = len(wordBag)

    # force a little housekeeping:
    sentences = None
    corpusText = None

    processDirectory(sys.argv[1])

articleCounter = 0
corpusWordLength = 0
wordBag = []
EOSBag = []
main()
APPENDIX A. LIST OF PROGRAMS

A.3.3 Second Method

The second corpus randomizer creates a truly randomized corpus. It achieves this by following this process:

1. Tokenize the concatenated corpus by sentence.

2. For each sentence, strip it of its punctuation and tokenize it by words.
   Throw all words in a WordBag. Append a full-stop to the WordBag.

3. Then, for every article in the real corpus:
   - Perform a word-count.
   - Begin choosing words from the WordBag at random. For each proper word chosen, reduce the remaining word count by 1 and append it to the randomized article. For each full-stop, simply append it to the randomized article.

The script’s command-line usage is identical to that of the first randomizer script, see Code Listing A.15

```bash
#!/usr/bin/env python
# Corpus randomizer
# Create a corpus of the same number of articles as a 'real' corpus,
# each article having an identical word-count to the original article but comprising
# of random words.
##
# This is the totally random approach
##
# Aengus Walton <waltona@tcd.ie>
# April, 2009
##
#
```
import os, sys
from BeautifulSoup import BeautifulSoup
import nltk
from random import randint

def processDirectory(directory):
    global articleCounter
    dirs = [d for d in os.listdir(directory) if os.path.isdir(os.path.join(directory, d))]
    for d in dirs:
        processDirectory(os.path.join(directory, d))

    files = [f for f in os.listdir(directory) if f.endswith(".hist")]
    for f in files:
        articleCounter = articleCounter + 1
        print "Processing: #" + str(articleCounter)
        processArticle(directory, f[:-5] + ".txt")

def processArticle(directory, f):
    """
    Create a randomized article and save it
    """
    global corpusWords
    global corpusWordLength
    global wordBag

    remainingWordCount = getWordCount(os.path.join(directory, f))

    if remainingWordCount == 0:
        print "PASS."
    else:
        # find an appropriate place to put the new article file, similar
        # to that found in the original corpus:
        level1 = str(directory)[len(str(directory))-5:len(str(directory))-3]
        level2 = str(directory)[len(str(directory))-2:]
        if not os.path.exists(os.path.join(sys.argv[2], level1)):
            os.mkdir(os.path.join(sys.argv[2], level1))
        if not os.path.exists(os.path.join(sys.argv[2], level1, level2)):
            os.mkdir(os.path.join(sys.argv[2], level1, level2))
newFilename = os.path.join(sys.argv[2], level1, level2, f)

newArticleList = []
currentEOS = []
delthis = 0
newArticle = ""

while remainingWordCount > 0:
    x = randint(0, len(wordBag)-1)
    if wordBag[x] == ".": 
        newArticle = newArticle[:-1] + ". "
    else:
        newArticle += str(wordBag[x]) + " "
    remainingWordCount = remainingWordCount - 1
    del wordBag[x]

outFile = open(newFilename,"w")
outFile.write(newArticle)
outFile.close()

def getWordCount(filename):
    inFile = open(filename,"r")
    lines = ""
    for line in inFile:
        lines += line
    soup = BeautifulSoup(lines)
    if soup.text == None or soup.text.contents[0] == "\n" or soup.text.contents[0][:8] == "REDIRECT" or soup.text.contents[0][:9] == "nREDIRECT":
        return 0
    else:
        words = nltk.word_tokenize(removePunctuation(soup.text.contents[0]))
        return len(words)

def removePunctuation(text):
    """
    Remove all exclamation marks, full stops, question marks, quotation marks,
    parentheses, colons, semi-colons and commas from a string and return the
    modified string.
    """
APPENDIX A. LIST OF PROGRAMS


```python
result = ""
for c in text:
    if c in '!.?();,':
        pass
    else:
        result += c
return result

def main():
    global corpusWordLength
    global wordBag
    global EOSBag
    if len(sys.argv) < 3:
        print "Corpus Text Randomizer"
        print "="*30
        print "Takes a corpus of texts, split on the basis of individual articles"
        print "and constructs a similar corpus of randomized text."
        print "Each article will have the same number of words as the original,"
        print "but a random number of words per sentence."
        print ""
        print "Usage:"
        print "randomizer.py [root dir of original corpus] [output dir of randomized corpus] [textfile containing complete original corpus]"
    else:
        # tokenize entire corpus .txt file:
        print "Reading in full corpus file and tokenizing sentences..."
        sentence_tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
        corpusText = ""
        for line in open(sys.argv[3]):
            corpusText += line
        sentences = sentence_tokenizer.tokenize(corpusText)
        print "Creating mixed bag of words and EOS-particles..."
        for sentence in sentences:
            words = nltk.word_tokenize(removePunctuation(sentence))
```

for word in words:
    wordBag.append(word)
    wordBag.append(".")

corpusWordLength = len(wordBag)

# force a little house-keeping:
sentences = None
corpusText = None

processDirectory(sys.argv[1])

articleCounter = 0
corpusWordLength = 0
wordBag = []

main()

Code Listing A.17: Second Corpus Randomizer (Totally Random), randomizer2.py

A.3.4 Randomized Corpus Statistics Generator

The main statistics script statistics.py (see section A.2.2) needed to be modified slightly in order to work with a randomized corpus. It searches for .txt files instead of .hist files, doesn’t try to parse the XML that is present in the original corpora, and doesn’t try to find edit count and author count statistics for each article.

In all other respects, it is identical.