Master's thesis

Using material from Internet as a Corpus

2009-09-23

Lund University
Faculty of Engineering, LTH
Department of Computer Science

Author: Christian Larsson
Supervisor: Pierre Nugues

In cooperation with Oribi AB

Abstract

The purpose of this master's thesis was to create a large corpus of the Swedish language by using the Internet as a source. There are only small corpora available today, and to bring language research forward, it is important to have a big Swedish corpus.

Using material from the Internet as a corpus requires working through a number of steps such as to

- Download large amount of web pages from the Internet
- Extract the text from the web pages
- Identify Swedish text, and remove all text that isn't Swedish.
- Divide the text into sentences

Conclusion: This thesis resulted in two corpora, one from letting a crawler download web pages, and one from parsing the Wikipedia XML dump. The Wikipedia dump is of high quality and contains 2 686 698 sentences, corresponding to 44 395 946 words. The corpus that were created by crawling resulted in a corpus with 8 342 918 sentences, corresponding to 119 500 499 words. The crawled corpus is considered to be of a lower quality since it may contain some entries with foreign text, and some sentences may not be real sentences. It was also created unigram, bigram, and trigram statistics where we looked at frequency statistics over occurrences of word sequences in the both corpora.

Sammanfattning

Syftet med detta examensarbete är att skapa en stor korpus över det svenska språket, med Internet som källa. Detta är nödvändigt eftersom det idag bara finns små korpusar tillgängliga, och för att kunna föra språkforskningen framåt är det viktigt att ha tillgång till en stor korpus.

Att bygga upp en korpus med Internet som källa innebär att arbete i flera olika steg, så som att

- Ladda hem stora mängder hemsidor från Internet
- Utvinna text från hemsidorna
- Känna igen svensk text och ta bort all text som inte är skriven på svenska
- Dela upp texten i meningar

Slutsats: Detta examensarbete resulterade i två korpusar, en från att låta en webbspindel ladda hem webbsidor, samt en från att parsa Wikipedias XML-dump. Wikipedia-dumpen anses vara av hög kvalité och innehåller 2 686 698 meningar, motsvarande 44 395 946 ord. Korpusen som erhölls från att webbspindeln resulterade i en korpus med 8 342 918 meningar, motsvarande 119 500 499 ord. Korpusen som erhölls från att crawla Internet anses vara av lägre kvalité eftersom den kan innehålla några poster med utländsk text, samt att det förekommer att de uppmärkta meningarna inte egentligen är meningar.
Det togs även fram unigram-, bigram- och trigramstatistik för de båda korpusarna där vi tittade på frekvensen för ordsekvenser i de båda korpusarna.

Contents

1 Introduction

A corpus is a large collection of samples of a language. Corpora are tools to study how a language is used in written or spoken language. A corpus may be designed in different ways depending on what purpose the corpus expects to serve, i.e. a corpus may contain various information about origin, context etc.

The purpose of this master's thesis was to create a corpus for the Swedish language by using Internet as a source. The goal was to create a high quality corpus containing sentences gathered from public web pages. We wanted to create a corpus of a respectable size compared to existing corpuses, which means about 100 million words.

This thesis was done in cooperation with Oribi AB (see ref [1]), who develops assistive software for people with reading and writing difficulties such as dyslexia. Oribis software aims to help people spell and write grammatically correctly as they write on different platforms. The software that Oribi develops needs large quantities of text in form of a corpus to do statistical analysis on to improve its performance and Oribis intentions is to use the result from this thesis for this.

The simplest ways to detect erroneous writing is by using a dictionary and compare every word to this. Using that approach won't give any correction for faulty grammar and neither can it know which word the author mean for similar words, such as the Swedish words *släckt* (light off) and *släkt (family)*. Because of the above reasons Oribi needs a corpus to be able to make better analysis.

Oribi requested a corpus where each line consists of a sentence. By examining such a corpus, it is possible to look at the environment in which a word occurs and from that decide if the spelling is correct. To make the analysis even better, it is often created n-gram statistics as well where the frequencies of all word sequences in the corpus can be studied.

1.1 Previous Work

Google has created a large n-gram corpus which is an inspiration for this project. The Google n-gram corpus is an English corpus which was created by using Internet as a source. Google has successfully created a corpus with 1,024,908,267,229 words from public Web pages.

This thesis will try to create something similar to the Google corpus for the Swedish language. For obvious reasons there is no way that this thesis can compete with a multi-billion dollar company as Google and create such a big corpus.

This thesis aims to create a corpus with about 100 million words, which is bigger than all the currently available Swedish corpora. Currently the biggest Swedish corpus is Parole, which contains about 20 million words. There is also a high quality corpus with about 1 million words, SUC (Stockholm Umeå Corpus), see ref[2]. The SUC corpus is smaller than Parole, but it contains more information, each word has been tagged, i.e. annotated with part-of-speech, inflectional form and lemma. For the English language there are among others the British National Corpus, which contains about 100 million words, and the Google corpus which has been discussed above.

1.2 Strategy

It was decided to create two corpora, one by using Wikipedia as a source, and one where we set up a crawler to crawl and download a wide range of pages.

Wikipedia is considered to be a high quality source of text due to the fact that it’s continuously being checked and corrected by users, compared to the web where everyone can write anything and the correctness can not be guaranteed. The Wikipedia corpus contains information about how the language is used in a correct way, while the web corpus gives information about how the language is being used in less formal situations. It is desirable to keep these two corpora separated because of the above reasons.

No crawl is necessary to get the content of Wikipedia since Wikipedia continuously creates dumps of its content in both HTML and XML format.

2 Crawling The Web

In order to use the web as a corpus, it was necessary to use a web crawler to download large amounts of web pages. A web crawler is a computer program that traverses the web in a methodical and automated way. Web crawlers are commonly used to gather and index web pages for search engines. They may also be used for different purposes such as harvesting e-mails or gathering other information.

For this project we used Nutch, which is an open source search engine within the Lucene project. The Nutch crawler offers many features and is highly configurable via plug-ins. For more reading about the Lucene project and Nutch, see ref [3].

2.1 Configuring the Crawler

2.1.1 Setting Up a Starting Point

The crawler requires at least one starting URL, which will work as the starting point for the crawler. From this point the crawler will browse to the entered site and parse the page for links. These links will be added to a database, and for the next fetch round the procedure will be repeated for all the pages that have been added to the database. The ideal condition for crawling the web would be if one starting address would lead the crawler around the whole web. This would imply that the web would be a connected graph, which it in fact isn't. To get a good crawl, it is important to have many starting addresses.

In this project the DMOZ open directory URL-list was used, containing 4 559 780 urls. DMOZ is claimed to be the largest, most comprehensive human-edited directory of the Web, see ref [4]. The list can be downloaded from:

http://rdf.dmoz.org/rdf/content.rdf.u8.gz

The occurrences of potential Swedish domains in the list were counted and it was found that the list contains 33 396 .se domains, and 4 265 .nu domains.

2.1.2 Which Pages to Fetch

For this project we were only interested in building a corpus over the Swedish language, and most of the pages on the Internet are not written in Swedish. A crawl without any restrictions might result in a crawl where only a few percent of the fetched pages would be useful. It was decided to use an approach where we try to aim the crawl at *.se* and *.nu* domains to get as much Swedish content as possible, and leave the rest. As mentioned earlier Nutch is open-source, and have many optional plug-ins, which can be used.
To filter which pages to fetch the *domain-urlfilter* was used. The *domain-urlfilter* is a plug-in which allows the user to define what domains to crawl.

The domain-urlfilter has been attached as **Appendix A**.

2.2 Running the Crawler

The Nutch crawler can be run in two modes: Intranet crawling and Whole-web crawling. The Intranet crawling is automated, while the whole-web crawling offers much greater control. A failed Intranet crawl can not easily be resumed if crashed, but as every step is controlled in the whole-web crawl, it is easy to resume a session that has crashed.

For this project a large crawl was proposed, and it was vital to have a fail-safe crawl. Therefore the whole-web crawl method was used.

The different steps in getting the crawler running are described below.

1. Create a database and inject the starting urls
   Command: `bin/nutch inject crawl/crawldb dmoz`
2. Generate a fetchlist of urls to be fetched in the next round
   Command: `/bin/nutch generate crawl/crawldb crawl/segments -topN`
3. Start the crawl
   Command: `bin/nutch fetch $segment -threads`
4. Update the database with the new urls
   Command: bin/nutch updatedb crawl/crawldb $segments
5. Go to step 2 for as many rounds as needed.

In step 2 the -topN option says how many urls to fetch in the specific segment. In step 3 the –threads option says how many simultaneous threads the crawler should be allowed to use while crawling.

In order to make the crawl automated a shell script was used. The shell-script that was used in this project was created with help of a tutorial provided by the Nutch community, see ref [5].
See **Appendix B** for the specific shell script used in this project.

2.3 Crawling With Politeness

During the preliminary test rounds, it was noticed that the crawler took long time at the end of each segment. This occurred because a segment may contain many URLs from the same domain, and the crawler has politeness setting, which doesn't allow it to fetch more than one page from the same domain at the same time. This problem could be solved by allowing the crawler to fetch 10 pages at a time from each domain. This solves the problem, but users at the Nutch discussion forum suggested that it's a bad idea and mean that it may end up with webmasters thinking that they are under attack and thus blocking the crawler. For this reason the number of allowed fetching threads per page should be kept at a low number, such as 1 or 2. The other way to solve this problem was by keeping the segments at a smaller size. If the crawler is told to fetch all pages in the database in each segment, it is obvious that many pages will be from the same domain, but if the segments are smaller, Nutch tries to divide the urls between the segments to avoid hammering of specific sites.

2.4 Merging the Segments

Our crawl resulted in 76 segments, which made it desirable to merge these into one segment for practical reasons.

Nutch offers a feature for merging segments into one big segment. However, this option has great performance disadvantages. While testing to merge the segments in this project it was noticed that the temporary folder used by Nutch grew out of control. While testing to merge some of the first segments the temp file grew to 70GB without any signs of stop growing.

This is a known problem with Nutch, and it has been widely discussed in the Nutch discussion forum and mail-list, see ref[6]. Users have reported that merging 8 segments with 250K urls each, made the temp directory grow to above 904GB. Therefore the built in merging utility was not used for this project.

As stated above, the Nutch merging feature was judged unsuitable for this why a manual merge was made instead.

To get the data from the segments, Nutch offers a dump feature, which gives the data in plain text. The command for this is:

```
bin/nutch readseg -dump crawl/segments/... output -nofetch -nogenerate
```

To do this for all segments the following command was used:

```
for A in `find -type d -mindepth 1 -maxdepth 1`; do ../bin/nutch
readseg -dump $A "$A-new" -nofetch -nogenerate; done
```

To merge the dumps the following command was used:

```
for A in `find -type d -mindepth 1 -maxdepth 1`; dot cat "$A/dump" >>
total; done
```

3 Extracting the Data

The code for extracting the data has been attached as Appendix:
Appendix D for Wikipedia
Appendix E for the crawled data

3.1 Wikipedia

Different options were considered for extracting the content from Wikipedia. One option was to download Wikipedia in HTML-format, and then parse the pages by using a HTML-parser. Another option was to download the Wikipedia XML-dump and parse the XML-document.

The benefits of using the HTML-dump is that the HTML-documents are well formed and all tags are known HTML-tags, which make it possible to use an already developed HTML-parser.

The XML-dump is also well formed with respect to the XML-tags and could also easily be parsed by using an XML-parser. The main problem is that the wiki-text in the XML document uses MediaWikis own markup language, which is not widely used, and therefore may be harder to parse.

It was found that there are tools available for parsing the Wikipedia XML-dump to plain text, since the primary goal of this thesis was to create a corpus, it was decided to use these tools and not try to create our own.

The parser that was chosen for this project was a Python script developed by Medialab, see ref [7]. The Python script was modified to fit the needs of this project, and the code used can be found in **Appendix C.**

The script parses the Wikipedia XML dump and extracts plain text. Originally the script stripped all the head markings and left only the text. The script was modified to keep information about all headings and at what level they were at. This was done because we wanted to create a corpus with as much information as possible. By keeping the headings, one can look at them, look at the text that is connected to a specific heading and draw conclusions from that.

3.1.1 The Wikipedia Markup Language

Wikipedia is an open source project which has developed its own markup language. This markup language is developed by the Wikipedia community and does not follow any other standards. The markup language is complex, and one can expect to encounter numerous pitfalls while trying to write such a parser.

Some examples of the Markup Language can be seen below:

> [[Öresund]]

The brackets indicate that the text should be considered as an internal link as well. This line creates a link to the Wikipedia article about "Öresund".

> [[Danmark|dansk]]

This is basically the same tag, but one also has the option to make the text in the article say one thing, and the link goes somewhere else. In this example the text says "dansk", and the link points to the wikiarticle about "Danmark".

A more complex example of how the markup language can be used is shown below.

```
{{Stapeldiagram
|titel=Befolkningsutvecklingen i Abbekås
|titelfärg=#DDD
|bredd=400px
|diagrambredd=300px
|vänster1=År
|höger1=Invånare
|staplar=
{{Pixelstapel|1990|#0099FF|63|4}}
{{Pixelstapel|1995|#0099FF|68|3}}
{{Pixelstapel|2000|#0099FF|67|4}}
{{Pixelstapel|2005|#0099FF|70|8}}
|beskrivning=Källa:
[http://www.ssd.scb.se/databaser/makro/Visavar.asp?yp=tansss&xu=C
9233001&huvudtabell=FolkmangdTatort&deltabell=1&deltabellnamn=Fol
km%E4ngden+per+t%E4tort%2E+Vart+femte+%E5r&omradekod=BE&omradetex
t=Befolkning&preskat=O&innehall=Folkmangd&starttid=1990&stopptid=
2005&Prodid=BE0101&fromSok=Sok&Fromwhere=S&lang=1&langdb=1  SCB  -
Folkmängden per tätort. Vart femte år 1990-2005].
}}
```

This code results in the following diagram picture.

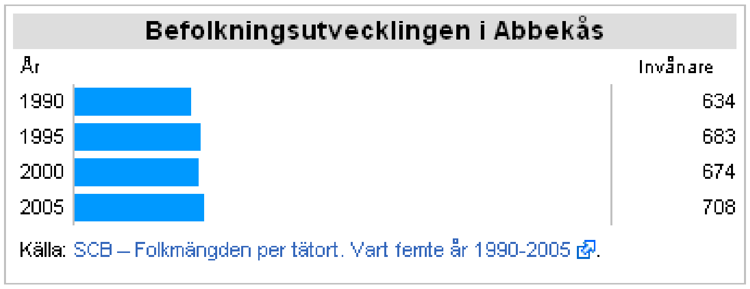

Figure 3.1: Example of a diagram created by the Wikipedia Markup Language

For a complete definition of the Wikipedia Markup Language see ref [8].

3.2 The Crawled Data

The Nutch crawler downloads and stores all html pages in one file per segment. The dump contains the source code of the webpage, and some additional metadata. Nutch has the option to parse the text on each page and include this in the dump, which we used.

The Nutch parser did a good job extracting the text. As an alternative, it was tried to manually parse the text by using the HTMLEditorKit in Java, but it did not give a better result than the one Nutch provided.

For the Wikipedia corpus, it was chosen to save the headings as they occur in the text, and it was discussed if something similar could be done in the crawled corpus. The text that was parsed from Nutch does not contain the headings, but the headings can be parsed from the HTML content separately. However, the situation is much more complex with the crawled data than in the Wikipedia material.

When browsing through the parsed text from the crawled data it was found that Nutch parses all text on a webpage, which means that text from tables, lists etc must be recognized and ignored. In this process it is unavoidable that some quality text gets thrown away as well. Trying to connect the different headings with specific text after this process is complicated, or even impossible. It is also rare that web pages actually contain big chunks of text divided by headings. As the headings were parsed and reviewed, it was found that they often contain rather useless information. Examples of frequently used headings would be: “Welcome”, “Information”, “Contact us” etc. Because of these reasons it was chosen to parse the headings, but present them after the sentences in the crawled corpus. Since Nutch only extract the text we had to use a HTML parser and parse the headings separately. This is done by using the HTMLEditorKit in java, and the code for doing this can be found in **Appendix E** (see class TagStripper).

3.3 Sentence Boundary Detection

The goal was to design a corpus containing one sentence per line. To do this an automatic way to divide a long text into sentences must be designed. The most obvious way to look for where a sentence starts or ends is by looking for the "." sign. This criteria isn't enough since the "." sign is ambiguous with respect to sentence boundary. There are several examples where the "." sign does not indicate the start of a new sentence, as in abbreviations such as *U.S*, or *e.g.* The later example is tricky since it ends with a dot and is followed by a whitespace, which is easily confused with the start of a new sentence. For more reading about sentence boundary detection see ref[9] and ref [10].

Java supports segmentation of text into sentences by using the class java.text.BreakIterator. The BreakIterator takes a local as an argument, which means that you specify which language the text is written in, and use that information to segment the text into sentences. Java code of how a text can be segmented into sentences is shown below.

```
Locale currentLocale = new Locale ("sv","SE");
BreakIterator sentenceIterator =
BreakIterator.getSentenceInstance(currentLocale);
sentenceIterator.setText(text);
int boundary2 = sentenceIterator.first();
int boundary1 = 0;

while (boundary2 != BreakIterator.DONE) {
      System.out.println(text.substring(boundary1,boundary2));
      boundary1 = boundary2;
      boundary2 = sentenceIterator.next();
}
```

This way of detecting sentence boundaries works well. It handles abbreviations and other ambiguities with a good success rate.

3.4 Rules for Validating Sentences

Wikipedia is a high quality source of text, and all text that was extracted from Wikipedia is assumed to be well formed and of interest to include in the corpus.

When dealing with the text from the crawled data this is not the case. Text that has been parsed from a website can be ill formed, it may e.g. just be scattered words. A filter was designed to remove junk and only pass through relevant sentences. The filter was developed through empirical tests where a rule was added, and the outcome was observed.

Different sets of rules were tried, some more aggressive than others. It was discovered that if the filter was made aggressive enough, it was possible to get a corpus with high quality. The set back of using a strict filter is that many sentences are erroneously thrown away. The goal with this corpus is to study how language is used in "real life", this means that the filter can’t be too aggressive. It was also argued that it is better to pass through too much information in the first version of the corpus so that anyone who wants to use it can customize it for their own interests.

The following rules were implemented:

1. A sentence can not be empty
2. A sentence is not allowed to have two consecutive spaces
3. A sentence must contain at least one space
4. A sentence is not allowed to have a hanging dot with spaces on both sides (“ . “)
5. A sentence may not contain “.se”, “.com”, “.nu”, “.org”, “.net” or “www”
6. Only the following characters are allowed in a sentence
    a. Letters
    b. Digits
    c. .
    d. ,
    e. ?
    f. !
    g. &
    h. Space mark
    i. (
    j.)
    k. –
    l. “
    m. :
    n. ;
    o. /
    p. \
    q. “

These rules were created by empirical testing. Some of the rules are self-explaining, while others may seem illogical. Rule number 2 says that a sentence can’t contain two consecutive spaces. It would actually be desired to have sentences in the corpus where people faulty have inserted two spaces after each other, but because of the nature of the parsed text this was also a great indicator of junk text. Examples of sentences that have been thrown away because of this rule are:

```
<S> Registrera     ? </S>
<S> - Tipsa en vän, klicka här              Spara som favorit
NYHET! </S>
<S>  Homepage:     http://www.thangorodrim.net/  Older  Versions:
3.0.5 (843k), 3.0.4 (774k), 3.0.3 (773k), 3.0.0 (793k) </S>
```

Rule number 4 which says that a sentence can not contain a hanging dot results in a filter for the following type of sentences:

```
<S> THE STORY OF WHAT HAPPENED IN KARLSTAD IS HERE . ...and some
mugshots from Malmoe..!!! </S>
<S> Har også redigert litt på siden til Isak sin familie . 27.5:
Bloggen er oppdatert! </S>
```

The hanging dot also produced some problems with sentence boundary detection in terms of that a hanging dot is sometimes wrongly used by the author to indicate the end of a sentence, but not always.

Other rules that could be implemented if a cleaner corpus is desired is to remove all sentences that contain ":" , ";" or consecutive dots such as "..". These rules were not implemented in the corpus because it was argued that these signs are frequently used in writing and should therefore be in the corpus. However, those signs are also sometimes an indicator of junk text.

We wanted the text in the corpus to follow the rules that were admitted with respect to the Swedish spelling reform 1906, see ref [11].
To achieve this, text was disregarded if it contained any of the following substrings:

hv, äro, gingo, gåfvor, gåfva, hafva, fingo, voro, blevo, bedrevo, däröfver, ifver, blefvo, blifva, lofva, blifver, lefver, afrätta, kufva, afskära, afrätta, afsky.

If any of these words occurred on a page it was thrown away. The filter doesn't contain all possible keywords, but if one word is found the surrounding text should probably be disregarded as well.

The first rule checks for the specific occurrence of " hv", a space mark followed by hv. This rule both helps where the language identification has failed and lets Danish and Norwegian text through and it finds many words that were changed in the spelling reform such as: hvad, hvadan, hval, hvalf etc.

4 Language Identification

Since the purpose of this project was to create a Swedish corpus, it was crucial to find a way to identify which language a given text is written in. The Wikipedia corpus was assumed to be written in Swedish, but for the crawled data there was a considerable mix of different languages.

The language identification was done by using the Java package *org.knallgrau.utils.textcat.TextCategorizer*, see ref [12]. This package can categorize text and identify which language a given text is written in. The text categorizer works better the more text that is presented to it at a time. A short text have lower success rate than larger quantities of text. Languages that are similar such as Danish, Norwegian and Swedish, can be mixed up by the categorizer if not enough text is provided.

A web page may contain text in more than one language, tough it doesn't occur frequently. In an attempt to handle this, it was tried to divide the text into sentences and then categorize each sentence. As stated above, the categorizer doesn't work well with short sentences, which led to many faulty identifications. To solve this, it was tried to set different thresholds that had to be fulfilled to let a text pass through, such as if 70% of the sentences are recognized as Swedish, the others are let through as well.

This way of handling the text works well, but the categorizing of text was found to be slow. It would take about 3+ weeks to parse all crawled data if every sentence was to be categorized.

To improve the performance of the program, the way of categorizing the text was changed. It was tried to put all sentences from each webpage together into a long text and then categorize everything at once. This was successful, there were no noticeable performance difference with respect to what text was outputted as Swedish, but the performance gain in time was great. Compared to 3+ weeks, it now only took 4 days to parse through the data.

5 Publishing the Result

This thesis resulted in two corpora that were aimed to be available for anyone to use and improve as long as its origin is presented. This makes it important to publish the result in a standardized way which is easy to understand and to parse. It was chosen to use XML syntax since it's a common way to publish this type of data.

5.1 Format

5.1.1 The Wikipedia Corpus

The following information was considered important to the Wikipedia corpus:

- An id number that increase with every article
- An URL to the current article (because of the dynamics of Wikipedia the text on the web may not always be the same as the text in the corpus)
- The title of the article
- The sentences
- The headings

This resulted in a corpus with the following format.

```
<doc id="#" url = "http://sv.wikipedia.org/wiki/...">
<Title>Example of a Title</Title>
<S> This is a sentence </S>
        …..
<H1>Heading on level 1</H1>
<S> This is another sentence </S>
        …..
<H2>Heading on level 2</H2>
<S> This is a third sentence </S>
        …..
</doc>
```

An example of what the corpus entry for the first article in the Wikipedia corpus look like has been attached as **Appendix F**.

5.1.2 The Crawled Corpus

The crawled corpus has a similar format as the one used for the Wikipedia corpus. In the crawled corpus, there is no Title tag, and the headings are not mixed with the text. The crawled corpus has the following format:

```
<doc id=”#” url = “http://...”>
<S>This is a sentence</S>
<S>This is another sentence</S>
<S>This is a third sentence</S>
          …..
<H1>Heading on level 1</H1>
<H1>Heading on level 1</H1>
<H2>Heading on level 2</H2>
          …..
</doc>
```

6 N-gram Statistics

It was desirable to get statistics, which shows the frequency of the words and sequences of words in a corpus to be able to draw conclusions of the fact that some words are more probable to follow each other than others. This type of statistics is called n-gram statistics. The unigram statistics is a frequency table of the tokens in the corpus, the bigram statistics is a frequency table of every 2-token sequence in the corpus, trigram, fourgram etc is the same for every 3-token and 4-token sequence.

The n-gram statistics presented below was created with a threshold of 10 occurrences, which means that no sequence that occurs less than 10 times was included in the n-gram statistics. When creating the n-gram statistics all headings were stripped out of the two corpora and only sentences were taken in consideration.

The statistics that are presented in this section can be compared to the n-gram statistics Google has published on the web, see ref [13].

The Lua script used to create the 3-gram statistics has been attached as **Appendix G**.

The counts are as follows:

6.1 Wikipedia

Key figures of this corpus:

Number of tokens:	52 468 401
Number of sentences:	2 686 698
Number of unigrams:	188 346
Number of bigrams:	426 609
Number of trigrams:	757 226

The 20 most frequent tokens in the unigram statistics:

Rank	Word	Frequency	Rank	Word	Frequency
1	`<S>`	2686698	11	på	530644
2	`</S>`	2686698	12	till	528045
3	.	2627810	13	med	517729
4	och	1494132	14	den	445973
5	i	1421304	15	för	442561
6	av	852581	16	var	332645
7	en	827240	17	ett	311400
8	som	812248	18	har	280306
9	är	688215	19	det	277774
10	att	608211	20	de	276176

Table 6.1: The 20 most frequent tokens in the unigram statistics in the Wikipedia corpus

The 20 most frequent 2-token sequences in the bigram statistics:

Rank	Sequence	Frequency	Rank	Sequence	Frequency
1	`. </S>`	2627806	11	`<S> En`	50700
2	`är en`	162315	12	`<S> Under`	43979
3	`<S> Han`	121713	13	`bland annat`	41123
4	`<S> I`	116756	14	`av de`	36864
5	`<S> Den`	109900	15	`<S> Efter`	36792
6	`för att`	101055	16	`i en`	35805
7	`<S> Det`	90205	17	`en av`	34615
8	`<S> De`	59782	18	`<S> På`	32012
9	`är ett`	52097	19	`av en`	31788
10	`var en`	51552	20	`i den`	30773

Table 6.2: The 20 most frequent 2-token sequences in the bigram statistics in the Wikipedia corpus

The 20 most frequent 3-token sequences in the trigram statistics:

Rank	Sequence	Frequency	Rank	Sequence	Frequency
1	`<S> Han var`	20233	11	`<S> Det var`	7996
2	`<S> Det finns`	14739	12	`Stockholm . </S>`	7893
3	`<S> Det är`	14539	13	`<S> För att`	7608
4	`på grund av`	13924	14	`<S> Efter att`	7605
5	`en av de`	12005	15	`var en svensk`	7498
6	`en del av`	10856	16	`<S> Den är`	7397
7	`<S> Han har`	10695	17	`USA . </S>`	7363
8	`Sverige . </S>`	10430	18	`i Sverige .`	7170
9	`<S> Han är`	9702	19	`är en svensk`	7094
10	`kommun . </S>`	8035	20	`invånare . </S>`	6867

Table 6.3: The 20 most frequent 3-token sequences in the trigram statistics in the Wikipedia corpus

The following is an example of word sequences in the 3-gram corpus which contains the word “universitet”:

Rank	Sequence	Frequency
1	`universitet . </S>`	2498
2	`vid Uppsala universitet`	1166
3	`vid universitetet i`	1153
4	`vid Lunds universitet`	804
5	`Uppsala universitet .`	460
6	`vid Stockholms universitet`	451
7	`Lunds universitet .`	367
8	`vid Göteborgs universitet`	288
9	`universitetet . </S>`	278
10	`vid Helsingfors universitet`	256

Table 6.4: Example of trigrams containing the word *universitet* in the Wikipedia corpus

Below are two histograms which show N-gram statistics for the Wikipedia corpus. The x-axis represents the count of how many times a sequence occurs in the n-gram statistics, and the y-axis represents how many sequences there are for that specific count.

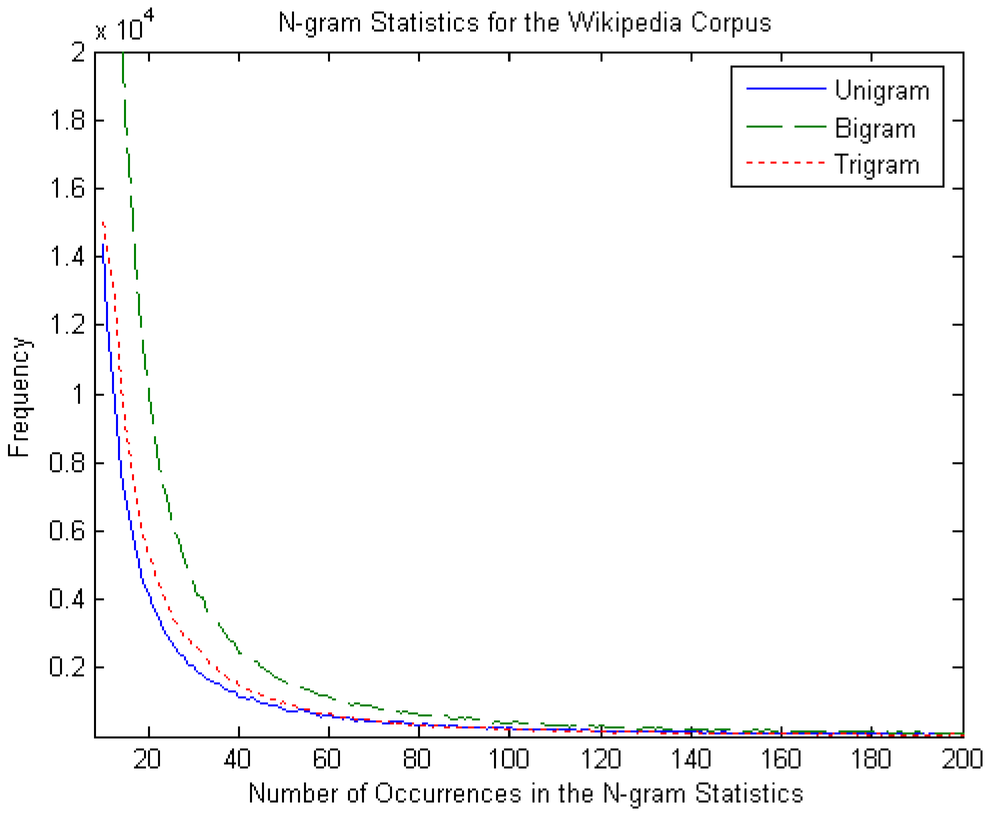

Figure 6.1: Diagram showing the relation between number of occurrences and frequency

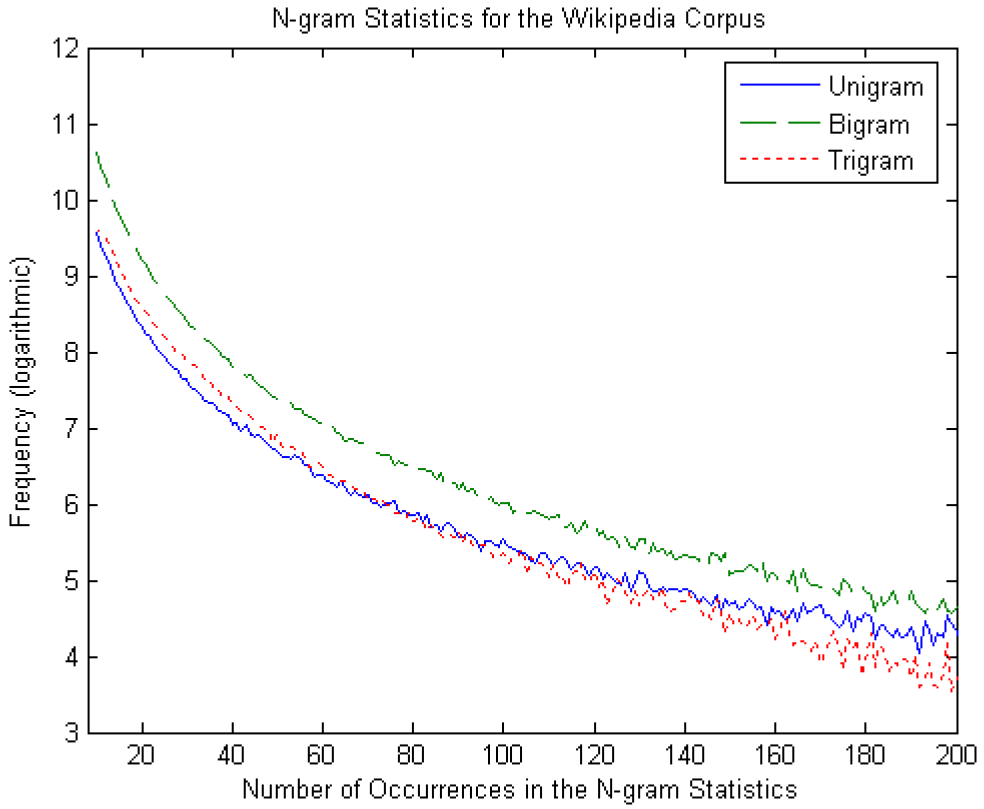

Figure 6.2: Diagram showing the relation between number of occurrences and frequency

6.2 Crawled Corpus

Key figures of this corpus:

Number of tokens:	145 679 852
Number of sentences:	8 342 918
Number of unigrams:	368 691
Number of bigrams:	1 104 584
Number of trigrams:	1 212 839

The 20 most frequent tokens in the unigram statistics:

Rank	Word	Frequency	Rank	Word	Frequency
1	`<S>`	8342918	11	för	1463478
2	`</S>`	8342918	12	med	1381048
3	.	6909467	13	av	1334560
4	och	3611218	14	till	1262162
5	i	2537131	15	det	1216796
6	att	2377097	16	har	930594
7	på	1825557	17	om	826339
8	är	1760129	18	du	745092
9	som	1757020	19	den	730948
10	en	1606788	20	ett	718228

Table 6.5: The 20 most frequent tokens in the unigram statistics in the crawled corpus

The 20 most frequent 2-token sequences in the bigram statistics:

Rank	Sequence	Frequency	Rank	Sequence	Frequency
1	`. </S>`	6907133	11	är det	124987
2	`! </S>`	649963	12	`<S>` En	122091
3	`? </S>`	613774	13	`<S>` Men	120390
4	`<S>` Det	378521	14	`<S>` Den	120192
5	för att	337540	15	det är	118146
6	`<S>` Vi	231952	16	att det	112313
7	`<S>` I	167549	17	`<S>` Läs	108563
8	`<S>` Jag	164310	18	`<S>` För	98519
9	är en	127282	19	`<S>` De	98213
10	Det är	125448	20	`<S>` Om	97853

Table 6.6: The 20 most frequent tokens in the bigram statistics in the crawled corpus

The 20 most frequent 3-token sequences in the trigram statistics:

Rank	Sequence	Frequency	Rank	Sequence	Frequency
1	`<S> Det är`	110862	11	`.. . </S>`	27517
2	`<S> Läs mer`	62488	12	`<S> Jag har`	27100
3	`<S> Vi har`	41270	13	`<S> Det var`	27040
4	`här . </S>`	36437	14	`att det är`	26574
5	`år . </S>`	35679	15	`oss . </S>`	21509
6	`<S> För att`	34909	16	`2 . </S>`	20953
7	`<S> Det finns`	34540	17	`Sverige . </S>`	20460
8	`<S> Om du`	33711	18	`för att få`	20254
9	`<S> Du kan`	33114	19	`på . </S>`	19624
10	`det . </S>`	28923	20	`här ! </S>`	19610

Table 6.7: The 20 most frequent tokens in the trigram statistics in the crawled corpus

The following is an example of word sequences in the 3-gram corpus which contains the word "universitet":

Rank	Sequence	Frequency
1	`universitet . </S>`	3209
2	`vid Umeå universitet`	835
3	`vid Uppsala universitet`	743
4	`universitet och högskolor`	730
5	`universitetet . </S>`	640
6	`vid Stockholms universitet`	587
7	`vid Lunds universitet`	523
8	`Uppsala universitet .`	500
9	`Umeå universitet .`	455
10	`Stockholms universitet .`	403

Table 6.8: Example of trigrams containing the word *universitet* in the crawled corpus

Below are two histograms which show N-gram statistics for the crawled corpus. The x-axis represents the count of how many times a sequence occurs in the n-gram statistics, and the y-axis represents how many sequences there are for that specific count.

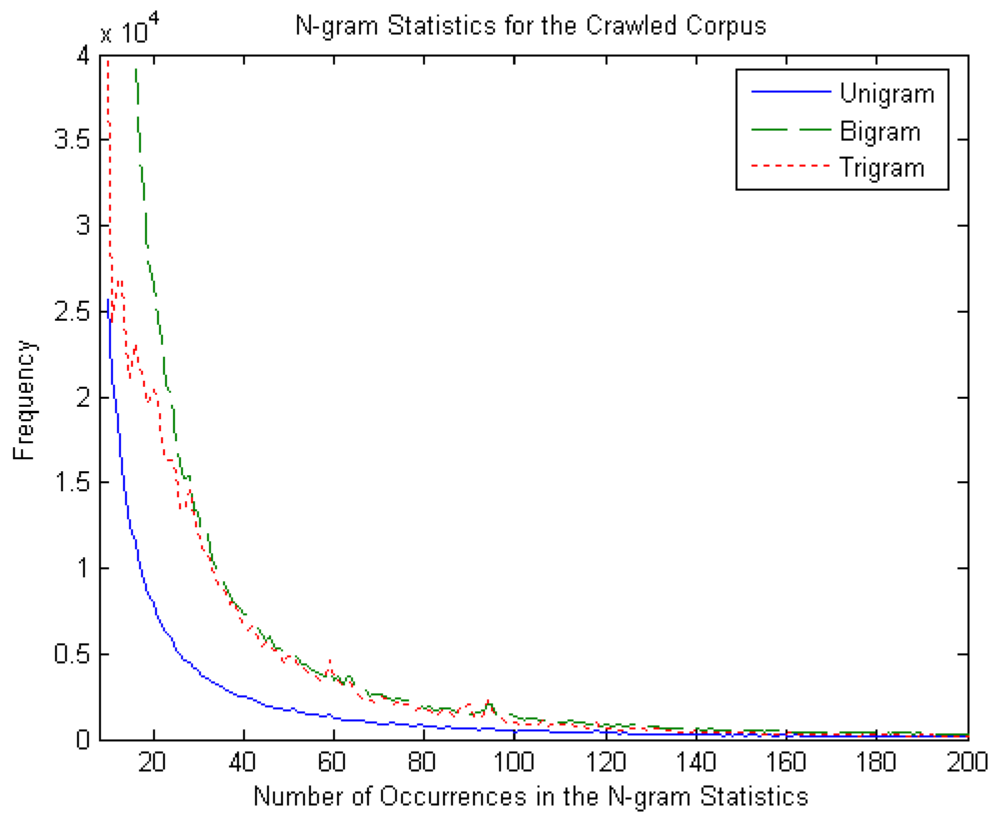

Figure 6.3: Diagram showing the relation between number of occurrences and frequency

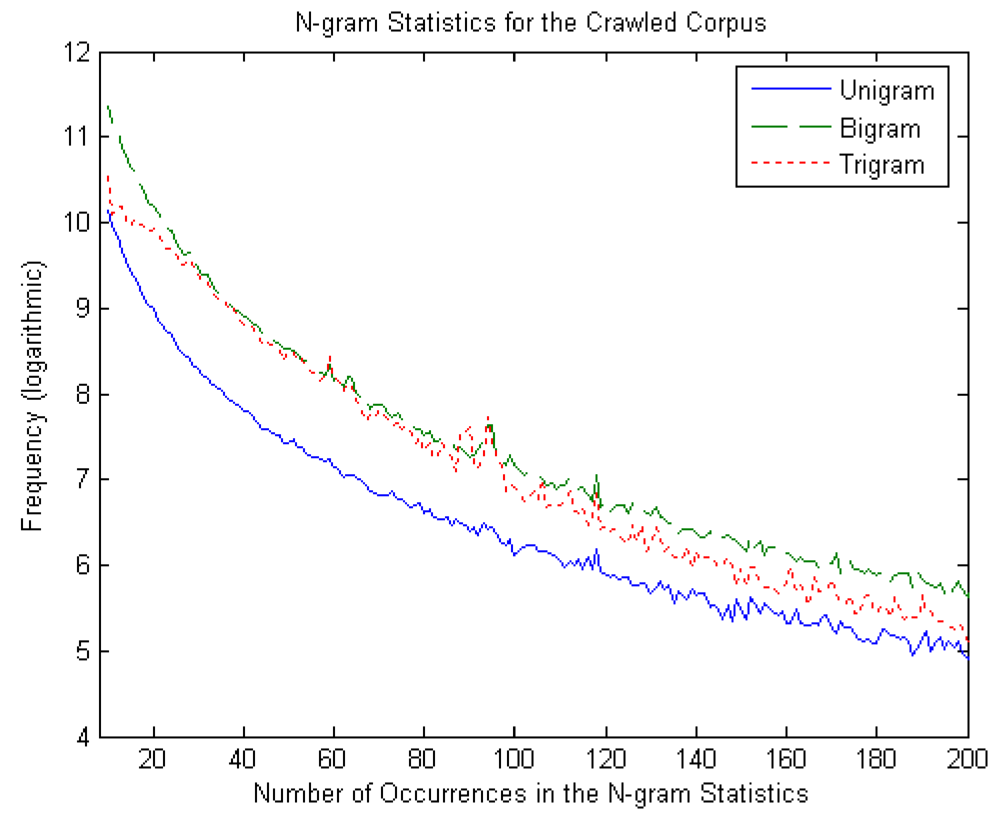

Figure 6.4: Diagram showing the relation between number of occurrences and frequency

7 Future Work and Improvements

For the Wikipedia corpus, it is possible to repeat the steps from this project to create corpora in other language that Wikipedia has been released in. There are no real difficulties in doing this, and the procedures described here are possible to repeat. Some languages may be special cases if they contain character sets that are not handled by the Python script, and the sentence boundary detection may not work properly for all languages.

For the crawled corpus there is always the option to try to create an even bigger corpus. More crawled data obviously means a bigger corpus. The major concern for an attempt to create a bigger corpus by crawling the whole web is that a lot of data will be downloaded, but only a small part of it will be Swedish. By looking at the dmoz url file one can conclude that only a couple of percents of the domains were Swedish. If this is assumed to be true for the web in general the text categorizing of such a crawl would not be realistic, since it would take about 50 times longer (200 days) to reach the same amount that was reached in this project.

The text in the crawled corpus has been categorized as Swedish, and even if the categorizer does a good job it was unavoidable that some foreign text passed through.

The filter that was designed to remove junk from the crawled text removed a lot of text that should not be in the corpus. The filter was designed in such a way that we wanted to let through as much real text as possible, and this came with the price that some text that shouldn't be in the corpus also slipped through, for example result lists from sport events and other types of listed content that may occur on websites.

8 The Result

The initial goal was to create a corpus with ~100 million words. The result was two corpora, one based on material from Wikipedia and one from crawling the Internet.

The Wikipedia corpus contains 2 686 698 sentences, corresponding to 44 395 946 words. From the crawled data, a corpus was created containing 8 342 918 sentences, corresponding to 119 500 499 words.

The project has successfully managed to gather corpora of such size and quality that it will be useful to the development of writing aids, but also many other applications within language technology.

9 Afterwords

The subject for this master's thesis was proposed by the company Oribi. I would like to thank all the personnel at Oribi for this opportunity, and especially Anders Holtsberg and Caroline Willners for giving directions and many valuable comments through out my work. I would also like to thank my supervisor from LTH, Pierre Nugues who has helped a lot to make this thesis a success.

10 References

[1] *Oribi, the company this Masters thesis was created in cooperation with.* *http://www.oribi.se*

[2] *The Stockholm Umeå Corpus SUC: Ejerhed, Källgren* *http://www.ling.su.se/staff/sofia/suc/suc.html*

[3] *The Lucene project* *http://lucene.apache.org/*

[4] *The Open Directory Project DMOZ* *http://www.dmoz.org/*

[5] *Nutch 0.9 Crawl Script Tutorial.* *http://wiki.apache.org/nutch/Nutch_0.9_Crawl_Script_Tutorial*

[6] *Nutch forum describing problems with segment merging.* *http://www.nabble.com/Merge-taking-forever-td23861788.html*

[7] *Medialab, developer of parsing tool for Wikipedia.* *http://medialab.di.unipi.it/wiki/Wikipedia_Extractor*

[8] *Definition of the MediaWiki Markup Language.* *http://www.mediawiki.org/wiki/Markup_spec*

[9] *"Sentence Boundary Detection Using a MaxEnt Classifier", 2005 Neha Agarwal, Kelley Herndon Ford, and Max Shneider, Department of Computer Science Stanford University, Stanford CA* *http://nlp.stanford.edu/courses/cs224n/2005/agarwal_herndon_shneider_final.pdf*

[10] *"Experiments on Sentence Boundery Detection", Mark Stevenson and Robert Gaizauskas, Department of Computer Science, University of Sheffield* *http://www.aclweb.org/anthology/A/A00/A00-1012.pdf*

[11] *Description of the Swedish spelling reform from 1906.* *http://sv.wikipedia.org/wiki/Stavningsreformen*

[12] *Java Text Categorizing Library.* *http://textcat.sourceforge.net/*

[13] *Information about the Google n-gram corpus.* *http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html*

Appendix A Domain-URLFilter Used by Nutch

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version
2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# config file for urlfilter-domsin plugin

se
nu
```

Appendix B *Shellscript for Running Nutch*

```
NUTCH_HOME=./
CATALINA_HOME=/var/lib/tomcat5.5

# Parse arguments
if [ "$1" == "safe" ]
then
  safe=yes
fi

if [ -z "$NUTCH_HOME" ]
then
  NUTCH_HOME=.
  echo runbot: $0 could not find environment variable NUTCH_HOME
  echo runbot: NUTCH_HOME=$NUTCH_HOME has been set by the script
else
  echo runbot: $0 found environment variable NUTCH_HOME=$NUTCH_HOME
fi

if [ -z "$CATALINA_HOME" ]
then
  CATALINA_HOME=/opt/apache-tomcat-6.0.10
  echo runbot: $0 could not find environment variable NUTCH_HOME
  echo runbot: CATALINA_HOME=$CATALINA_HOME has been set by the script
else
  echo runbot: $0 found environment variable CATALINA_HOME=$CATALINA_HOME
fi

if [ -n "$topN" ]
then
  topN="--topN $rank"
else
  topN=""
fi

steps=10
echo "----- Inject (Step 1 of $steps) -----"
$NUTCH_HOME/bin/nutch inject crawl/crawldb dmoz
echo "----- Generate, Fetch, Parse, Update (Step 2 of $steps) -----"
for((i=0; i < $depth; i++))
do
  echo "--- Beginning crawl at depth `expr $i + 1` of $depth ---"
  $NUTCH_HOME/bin/nutch generate crawl/crawldb crawl/segments -topN 50 000
  if [ $? -ne 0 ]
  then
    echo "runbot: Stopping at depth $depth. No more URLs to fetch."
    break
  fi
  segment=`ls -d crawl/segments/* | tail -1`

  $NUTCH_HOME/bin/nutch fetch $segment -threads $threads
  if [ $? -ne 0 ]
  then
    echo "runbot: fetch $segment at depth $depth failed. Deleting it."
    rm -rf $segment
    continue
  fi

  echo "--- Parsing Segment $segment ---"
  $NUTCH_HOME/bin/nutch parse $segment

  $NUTCH_HOME/bin/nutch updatedb crawl/crawldb $segment
done
```

Appendix C Python Script to Parse Wikipedia XML

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# =============================================================================
#  Version: 1.0 (Mar 12, 2009)
#  Author: Antonio Fuschetto (fuschett@di.unipi.it), University of Pisa
# =============================================================================

# =============================================================================
#  This file is part of Tanl.
#
#  Tanl is free software; you can redistribute it and/or modify it
#  under the terms of the GNU General Public License, version 3,
#  as published by the Free Software Foundation.
#
#  Tanl is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program.  If not, see <http://www.gnu.org/licenses/>.
# =============================================================================

"""Wikipedia Extractor:
Extracts and cleans text from Wikipedia database dump and stores output in a
number of files of similar size in a given directory. Each file contains
several documents in Tanl document format.

Usage:
  WikiExtractor.py [options]

Options:
  -c, --compress        : compress output files using bzip2 algorithm
  -b ..., --bytes=...   : put specified bytes per output file (500K by default)
  -o ..., --output=...  : place output files in specified directory (current
                          directory by default)
  --help                : display this help and exit
  --usage               : display script usage
"""

import sys
import getopt
import pickle
import urllib
import re
import bz2
import os.path

### SUPPORT CLASSES ###########################################################

class WikiDocument:
    def __init__(self):
        self.id = None
        self.url = None
        self.text = None

    def __str__(self):
        return '<doc id="%d" url="%s">\n%s\n</doc>\n' % (self.id, self.url, self.text)

#------------------------------------------------------------------------------

class WikiExtractor:
    __garbage_tags = ('ref', 'gallery', 'timeline', 'noinclude', 'pre', 'table', 'tr', 'td',
                      'ul', 'li', 'ol', 'dl', 'dt', 'dd', 'menu', 'dir')
    __wrapper_tags = ('nowiki', 'cite', 'source', 'hiero', 'div', 'font', 'span', 'strong',
                      'strike', 'blockquote', 'tt', 'var', 'sup', 'sub', 'big', 'small',
                      'center', 'h1', 'h2', 'h3', 'em', 'b', 'i', 'u', 'a', 's', 'p')
    __single_tags = ('references', 'ref', 'img', 'br', 'hr', 'li', 'dt', 'dd')
    __placeholder_tags = {'math':'Formula', 'code':'Codice'}

    __project_namespaces = ('wikipedia', 'mediawiki', 'wikiquote', 'wikibooks', 'wikisource',
                            'wiktionary', 'wikispecies', 'wikinews', 'wikiversita',
'commons')
    __garbage_namespaces = ('immagine', 'image', 'categoria', 'category')

    __char_entities =   {'&nbsp;'   :u'\u00A0', '&iexcl;' :u'\u00A1', '&cent;'    :u'\u00A2',
                        '&pound;'  :u'\u00A3', '&curren;':u'\u00A4', '&yen;'     :u'\u00A5',
                        '&brvbar;' :u'\u00A6', '&sect;'  :u'\u00A7', '&uml;'     :u'\u00A8',
                        '&copy;'   :u'\u00A9', '&ordf;'  :u'\u00AA', '&laquo;'   :u'\u00AB',
                        '&not;'    :u'\u00AC', '&shy;'   :u'\u00AD', '&reg;'     :u'\u00AE',
                        '&macr;'   :u'\u00AF', '&deg;'   :u'\u00B0', '&plusmn;'  :u'\u00B1',
                        '&sup2;'   :u'\u00B2', '&sup3;'  :u'\u00B3', '&acute;'   :u'\u00B4',
                        '&micro;'  :u'\u00B5', '&para;'  :u'\u00B6', '&middot;'  :u'\u00B7',
                        '&cedil;'  :u'\u00B8', '&sup1;'  :u'\u00B9', '&ordm;'    :u'\u00BA',
                        '&raquo;'  :u'\u00BB', '&frac14;':u'\u00BC', '&frac12;'  :u'\u00BD',
                        '&frac34;' :u'\u00BE', '&iquest;':u'\u00BF', '&Agrave;'  :u'\u00C0',
                        '&Aacute;' :u'\u00C1', '&Acirc;' :u'\u00C2', '&Atilde;'  :u'\u00C3',
                        '&Auml;'   :u'\u00C4', '&Aring;' :u'\u00C5', '&AElig;'   :u'\u00C6',
                        '&Ccedil;' :u'\u00C7', '&Egrave;':u'\u00C8', '&Eacute;'  :u'\u00C9',
                        '&Ecirc;'  :u'\u00CA', '&Euml;'  :u'\u00CB', '&Igrave;'  :u'\u00CC',
                        '&Iacute;' :u'\u00CD', '&Icirc;' :u'\u00CE', '&Iuml;'    :u'\u00CF',
                        '&ETH;'    :u'\u00D0', '&Ntilde;':u'\u00D1', '&Ograve;'  :u'\u00D2',
                        '&Oacute;' :u'\u00D3', '&Ocirc;' :u'\u00D4', '&Otilde;'  :u'\u00D5',
                        '&Ouml;'   :u'\u00D6', '&times;' :u'\u00D7', '&Oslash;'  :u'\u00D8',
                        '&Ugrave;' :u'\u00D9', '&Uacute;':u'\u00DA', '&Ucirc;'   :u'\u00DB',
                        '&Uuml;'   :u'\u00DC', '&Yacute;':u'\u00DD', '&THORN;'   :u'\u00DE',
                        '&szlig;'  :u'\u00DF', '&agrave;':u'\u00E0', '&aacute;'  :u'\u00E1',
                        '&acirc;'  :u'\u00E2', '&atilde;':u'\u00E3', '&auml;'    :u'\u00E4',
                        '&aring;'  :u'\u00E5', '&aelig;' :u'\u00E6', '&ccedil;'  :u'\u00E7',
                        '&egrave;' :u'\u00E8', '&eacute;':u'\u00E9', '&ecirc;'   :u'\u00EA',
                        '&euml;'   :u'\u00EB', '&igrave;':u'\u00EC', '&iacute;'  :u'\u00ED',
                        '&icirc;'  :u'\u00EE', '&iuml;'  :u'\u00EF', '&eth;'     :u'\u00F0',
                        '&ntilde;' :u'\u00F1', '&ograve;':u'\u00F2', '&oacute;'  :u'\u00F3',
                        '&ocirc;'  :u'\u00F4', '&otilde;':u'\u00F5', '&ouml;'    :u'\u00F6',
                        '&divide;' :u'\u00F7', '&oslash;':u'\u00F8', '&ugrave;'  :u'\u00F9',
                        '&uacute;' :u'\u00FA', '&ucirc;' :u'\u00FB', '&uuml;'    :u'\u00FC',
                        '&yacute;' :u'\u00FD', '&thorn;' :u'\u00FE', '&yuml;'    :u'\u00FF',
                        '&fnof;'   :u'\u0192', '&Alpha;' :u'\u0391', '&Beta;'    :u'\u0392',
                        '&Gamma;'  :u'\u0393', '&Delta;' :u'\u0394', '&Epsilon;' :u'\u0395',
                        '&Zeta;'   :u'\u0396', '&Eta;'   :u'\u0397', '&Theta;'   :u'\u0398',
                        '&Iota;'   :u'\u0399', '&Kappa;' :u'\u039A', '&Lambda;'  :u'\u039B',
                        '&Mu;'     :u'\u039C', '&Nu;'    :u'\u039D', '&Xi;'      :u'\u039E',
                        '&Omicron;':u'\u039F', '&Pi;'    :u'\u03A0', '&Rho;'     :u'\u03A1',
                        '&Sigma;'  :u'\u03A3', '&Tau;'   :u'\u03A4', '&Upsilon;' :u'\u03A5',
                        '&Phi;'    :u'\u03A6', '&Chi;'   :u'\u03A7', '&Psi;'     :u'\u03A8',
                        '&Omega;'  :u'\u03A9', '&alpha;' :u'\u03B1', '&beta;'    :u'\u03B2',
                        '&gamma;'  :u'\u03B3', '&delta;' :u'\u03B4', '&epsilon;' :u'\u03B5',
                        '&zeta;'   :u'\u03B6', '&eta;'   :u'\u03B7', '&theta;'   :u'\u03B8',
                        '&iota;'   :u'\u03B9', '&kappa;' :u'\u03BA', '&lambda;'  :u'\u03BB',
                        '&mu;'     :u'\u03BC', '&nu;'    :u'\u03BD', '&xi;'      :u'\u03BE',
                        '&omicron;':u'\u03BF', '&pi;'    :u'\u03C0', '&rho;'     :u'\u03C1',
                        '&sigmaf;' :u'\u03C2', '&sigma;' :u'\u03C3', '&tau;'     :u'\u03C4',
                        '&upsilon;':u'\u03C5', '&phi;'   :u'\u03C6', '&chi;'     :u'\u03C7',
                        '&psi;'    :u'\u03C8', '&omega;' :u'\u03C9', '&thetasym;':u'\u03D1',
                        '&upsih;'  :u'\u03D2', '&piv;'   :u'\u03D6', '&bull;'    :u'\u2022',
                        '&hellip;' :u'\u2026', '&prime;' :u'\u2032', '&Prime;'   :u'\u2033',
                        '&oline;'  :u'\u203E', '&frasl;' :u'\u2044', '&weierp;'  :u'\u2118',
                        '&image;'  :u'\u2111', '&real;'  :u'\u211C', '&trade;'   :u'\u2122',
                        '&alefsym;':u'\u2135', '&larr;'  :u'\u2190', '&uarr;'    :u'\u2191',
                        '&rarr;'   :u'\u2192', '&darr;'  :u'\u2193', '&harr;'    :u'\u2194',
                        '&crarr;'  :u'\u21B5', '&lArr;'  :u'\u21D0', '&uArr;'    :u'\u21D1',
                        '&rArr;'   :u'\u21D2', '&dArr;'  :u'\u21D3', '&hArr;'    :u'\u21D4',
                        '&forall;' :u'\u2200', '&part;'  :u'\u2202', '&exist;'   :u'\u2203',
                        '&empty;'  :u'\u2205', '&nabla;' :u'\u2207', '&isin;'    :u'\u2208',
                        '&notin;'  :u'\u2209', '&ni;'    :u'\u220B', '&prod;'    :u'\u220F',
                        '&sum;'    :u'\u2211', '&minus;' :u'\u2212', '&lowast;'  :u'\u2217',
                        '&radic;'  :u'\u221A', '&prop;'  :u'\u221D', '&infin;'   :u'\u221E',
                        '&ang;'    :u'\u2220', '&and;'   :u'\u2227', '&or;'      :u'\u2228',
                        '&cap;'    :u'\u2229', '&cup;'   :u'\u222A', '&int;'     :u'\u222B',
                        '&there4;' :u'\u2234', '&sim;'   :u'\u223C', '&cong;'    :u'\u2245',
                        '&asymp;'  :u'\u2248', '&ne;'    :u'\u2260', '&equiv;'   :u'\u2261',
                        '&le;'     :u'\u2264', '&ge;'    :u'\u2265', '&sub;'     :u'\u2282',
                        '&sup;'    :u'\u2283', '&nsub;'  :u'\u2284', '&sube;'    :u'\u2286',
                        '&supe;'   :u'\u2287', '&oplus;' :u'\u2295', '&otimes;'  :u'\u2297',
                        '&perp;'   :u'\u22A5', '&sdot;'  :u'\u22C5', '&lceil;'   :u'\u2308',
                        '&rceil;'  :u'\u2309', '&lfloor;':u'\u230A', '&rfloor;'  :u'\u230B',
                        '&lang;'   :u'\u2329', '&rang;'  :u'\u232A', '&loz;'     :u'\u25CA',
                        '&spades;' :u'\u2660', '&clubs;' :u'\u2663', '&hearts;'  :u'\u2665',
                        '&diams;'  :u'\u2666', '&quot;'  :u'\u0022', '&lt;'      :u'\u003C',
                        '&gt;'     :u'\u003E', '&OElig;' :u'\u0152', '&oelig;'   :u'\u0153',
                        '&Scaron;' :u'\u0160', '&scaron;':u'\u0161', '&Yuml;'    :u'\u0178',
                        '&circ;'   :u'\u02C6', '&tilde;' :u'\u02DC', '&ensp;'    :u'\u2002',
                        '&emsp;'   :u'\u2003', '&thinsp;':u'\u2009', '&zwnj;'    :u'\u200C',
                        '&zwj;'    :u'\u200D', '&lrm;'   :u'\u200E', '&rlm;'     :u'\u200F',
                        '&ndash;'  :u'\u2013', '&mdash;' :u'\u2014', '&lsquo;'   :u'\u2018',
                        '&rsquo;'  :u'\u2019', '&sbquo;' :u'\u201A', '&ldquo;'   :u'\u201C',
                        '&rdquo;'  :u'\u201D', '&bdquo;' :u'\u201E', '&dagger;'  :u'\u2020',
                        '&Dagger;' :u'\u2021', '&permil;':u'\u2030', '&lsaquo;'  :u'\u2039',
                        '&rsaquo;' :u'\u203A', '&euro;'  :u'\u20AC'}

    def __init__(self):
        # Riconosce i commenti HTML
        self.__comment_pattern = re.compile(r'<!--.*?-->', re.DOTALL)

        # Riconosce i tag HTML spazzatura
        self.__garbage_tag_patterns = list()
        for tag in self.__class__.__garbage_tags:
            pattern = re.compile(r'<\s*%s(\s*| [^/]+?)>.*?<\s*/\s*%s\s*>' % (tag, tag),
re.DOTALL | re.IGNORECASE)
            self.__garbage_tag_patterns.append(pattern)

        # Riconosce i tag HTML contenitori
        self.__wrapper_tag_patterns = list()
        for tag in self.__class__.__wrapper_tags:
            left_pattern = re.compile(r'<\s*%s(\s*| [^/]+?)>' % tag, re.DOTALL |
re.IGNORECASE)
            right_pattern = re.compile(r'<\s*/\s*%s\s*>' % tag, re.DOTALL | re.IGNORECASE)
            self.__wrapper_tag_patterns.append((left_pattern, right_pattern))

        # Riconosce i tag HTML singoli
        self.__single_tag_patterns = list()
        for tag in self.__class__.__single_tags:
            good_pattern = re.compile(r'<\s*%s(\s*| .+?)/\s*>' % tag, re.DOTALL |
re.IGNORECASE)
            bad_pattern = re.compile(r'<\s*(/|\\)?\s*%s(\s*| [^/]+?)\\?\s*>' % tag, re.DOTALL
| re.IGNORECASE)
            self.__single_tag_patterns.append((good_pattern, bad_pattern))

        # Riconosce i tag HTML segnaposto
        self.__placeholder_tag_patterns = list()
        for tag in self.__class__.__placeholder_tags.iterkeys():
            pattern = re.compile(r'<\s*%s(\s*| [^/]+?)>.*?<\s*/\s*%s\s*>' % (tag, tag),
re.DOTALL | re.IGNORECASE)
            self.__placeholder_tag_patterns.append((pattern,
self.__class__.__placeholder_tags[tag]))

        # Riconosce le tabelle e i template
        self.__table_pattern = re.compile(r'\{[^{]*?\}', re.DOTALL)

        # Riconosce i wikilink
        good_wikilink_pattern = re.compile(r'\[\[[^[]*?\]\]', re.DOTALL)
        bad_left_wikilink_pattern = re.compile(r'\[[^[]*?\]\]', re.DOTALL)
        bad_right_wikilink_pattern = re.compile(r'\[\[[^[]*?\]', re.DOTALL)
        self.__wikilink_pattern = (good_wikilink_pattern, bad_left_wikilink_pattern,
bad_right_wikilink_pattern)

        # Riconosce i link HTTP
        self.__http_link_pattern = re.compile(r'\[http.*?\]', re.DOTALL | re.IGNORECASE)

        # Riconosce gli apostrofi che precedono grassetto e corsivo
        apostrophe_bold_pattern = re.compile(r"\w'('''.*?''')", re.DOTALL)
        apostrophe_italic_pattern = re.compile(r"\w'(''.*?'')", re.DOTALL)
        self.__apostrophe_pattern = (apostrophe_bold_pattern, apostrophe_italic_pattern)

        # Riconosce le entita' numeriche
        self.__numeric_entity_pattern = re.compile(r'&#\d+?;')

        # Riconosce gli spazi multipli
        self.__multi_space_pattern = re.compile(r' {2,}')

        # Riconosce i punti multipli
        self.__multi_dot_pattern = re.compile(r'\.{4,}')

    def extract(self, wiki_document):
        wiki_document = self.__clean(wiki_document)
        if not wiki_document: return None

        wiki_document = self.__compact(wiki_document)
        return wiki_document

    def __clean(self, wiki_document):
        # Rende maggiormente riconoscibili i tag
        wiki_document.text = wiki_document.text.replace('&lt;', '<').replace('&gt;', '>')
        wiki_document.text = wiki_document.text.replace('<<', u'Â«').replace('>>', u'Â»')

        # Elimina i commenti HTML
        wiki_document.text = self.__comment_pattern.sub('', wiki_document.text)

        # Elimina i tag HTML spazzatura
        for pattern in self.__garbage_tag_patterns:
            wiki_document.text = pattern.sub('', wiki_document.text)

        # Elimina i tag HTML contenitori
        for left_pattern, right_pattern in self.__wrapper_tag_patterns:
            wiki_document.text = left_pattern.sub('', wiki_document.text)
            wiki_document.text = right_pattern.sub('', wiki_document.text)

        # Elimina i tag HTML singoli
        for good_pattern, bad_pattern in self.__single_tag_patterns:
            wiki_document.text = good_pattern.sub('', wiki_document.text)
            wiki_document.text = bad_pattern.sub('', wiki_document.text)

        # Elimina i tag HTML segnaposto
        for pattern, placeholder in self.__placeholder_tag_patterns:
            index = 1
            for match in pattern.finditer(wiki_document.text):
                wiki_document.text = wiki_document.text.replace(match.group(), '[%s %d]' %
(placeholder, index))
                index += 1

        # Elimina le tabelle e i template
        wiki_document.text = wiki_document.text.replace('{{end box}}', '}')
        wiki_document.text = wiki_document.text.replace('{{', '{').replace('}}', '}')
        wiki_document.text = wiki_document.text.replace('{|', '{').replace('|}', '}')
        wiki_document.text = self.__table_pattern.sub('', wiki_document.text)
        wiki_document.text = self.__table_pattern.sub('', wiki_document.text)
        wiki_document.text = self.__table_pattern.sub('', wiki_document.text)

        # Gestisce i wikilink (ben formattati)
        good_wikilink_pattern = self.__wikilink_pattern[0]
        for match in good_wikilink_pattern.finditer(wiki_document.text):
            wikilink = match.group()
            wiki_document.text = wiki_document.text.replace(wikilink,
self.__handle_wikilink(wikilink[2:-2]))
        for match in good_wikilink_pattern.finditer(wiki_document.text):
            wikilink = match.group()
            wiki_document.text = wiki_document.text.replace(wikilink,
self.__handle_wikilink(wikilink[2:-2]))

        # Gestisce i wikilink (mal formattatia)
        bad_left_wikilink_pattern = self.__wikilink_pattern[1]
        for match in bad_left_wikilink_pattern.finditer(wiki_document.text):
            wikilink = match.group()
            wiki_document.text = wiki_document.text.replace(wikilink,
self.__handle_wikilink(wikilink[1:-2]))
        bad_right_wikilink_pattern = self.__wikilink_pattern[2]
        for match in bad_right_wikilink_pattern.finditer(wiki_document.text):
            wikilink = match.group()
            wiki_document.text = wiki_document.text.replace(wikilink,
self.__handle_wikilink(wikilink[2:-1]))
        wiki_document.text = wiki_document.text.replace('[[', '').replace(']]', '')

        # Elimina i link HTTP
        wiki_document.text = self.__http_link_pattern.sub('',
wiki_document.text).replace('[]', '')

        # Gestisce i grassetti e i corsivi
        apostrophe_bold_pattern = self.__apostrophe_pattern[0]
        for match in apostrophe_bold_pattern.finditer(wiki_document.text):
            bold_text = match.group(1)
            wiki_document.text = wiki_document.text.replace(bold_text, bold_text[3:-3])
        apostrophe_italic_pattern = self.__apostrophe_pattern[1]
        for match in apostrophe_italic_pattern.finditer(wiki_document.text):
            italic_text = match.group(1)
            wiki_document.text = wiki_document.text.replace(italic_text, '&quot;%s&quot;' %
italic_text[2:-2])
        wiki_document.text = wiki_document.text.replace("'''", '').replace("''", '&quot;')

        # Gestisce i caratteri speciali
        wiki_document.text = wiki_document.text.replace('&amp;', '&').replace('&quot;&quot;',
'&quot;')
        for entity in self.__class__.__char_entities.iterkeys():
            wiki_document.text = wiki_document.text.replace(entity,
self.__class__.__char_entities[entity])

        # Gestisce i caratteri speciali
        for match in self.__numeric_entity_pattern.finditer(wiki_document.text):
            entity = match.group()
            wiki_document.text = wiki_document.text.replace(entity,
self.__handle_unicode(entity))

        # Gestisce alcune imperfezioni del testo
        wiki_document.text = wiki_document.text.replace('\t', ' ')
        wiki_document.text = self.__multi_space_pattern.sub(' ', wiki_document.text)
        wiki_document.text = self.__multi_dot_pattern.sub('...', wiki_document.text)
        wiki_document.text = wiki_document.text.replace(' ,', ',').replace(' .', '.')
        wiki_document.text = wiki_document.text.replace(' :', ':').replace(' ;', ';')
        wiki_document.text = wiki_document.text.replace(',,', ',').replace(',.', '.')
        wiki_document.text = wiki_document.text.replace('( ', '(').replace(' )', ')')
        wiki_document.text = wiki_document.text.replace('[ ', '[').replace(' ]', ']')
        wiki_document.text = wiki_document.text.replace(u'Â« ', u'Â«').replace(u' Â»', u'Â»')

        return wiki_document

    def __compact(self, wiki_document):
        page = list()
        paragraph = list()

        for line in wiki_document.text.split('\n'):
            line = line.strip()
            if not line: continue

            # Gestisce il titolo della pagina
            if line.startswith('++'):
                title = line[2:-2]
                if title and title[-1] not in '!?':
                    title = '%s.' % title
                page = [title]
            # Gestisce i titoli dei paragrafi
           # elif line.startswith('=='):
            #    if len(paragraph) > 1:
             #       page.extend(paragraph)
              #  title = line[2:-2]
              #  if title and title[-1] not in '!?':
              #      title = '%s.' % title
              #  paragraph = [title]
            # Elimina gli elenchi puntati e numerati
               # test
            elif line[0] in '*#:;':
                continue
            # Elimina i resti delle tabelle
            elif line[0] in '{|' or line[-1] in '}':
                continue
            # Elimina le righe non significative
            elif (line[0] == '(' and line[-1] == ')') or line.strip('.-') == '':
                continue
            # Elimina le righe con un basso numero di token
            elif len(line.split()) < 6:
                continue
            # Gestisce il testo della pagina
            elif len(paragraph) == 0:
                page.append(line)
            # Gestisce il testo dei paragrafi
            else:
                paragraph.append(line)

        if len(paragraph) > 1:
            page.extend(paragraph)
        elif len(page) == 1: return None
        wiki_document.text = '\n'.join(page)
        return wiki_document

    def __handle_wikilink(self, wikilink):
        tokens = wikilink.split(':')
        while not tokens[0]:
            if len(tokens) < 2: return ''
            tokens = tokens[1:]

        if len(tokens) == 1 or tokens[0].strip().lower() in
self.__class__.__project_namespaces:
            tokens = tokens[-1].split('|')
            while not tokens[-1]:
                if len(tokens) < 2: return ''
                tokens = tokens[:-1]
            return tokens[-1].split('#')[-1].split('/')[-1].strip()

        if tokens[0].strip().lower() in self.__class__.__garbage_namespaces: return ''

        tokens = tokens[-1].split('|')
        while not tokens[-1]:
            if len(tokens) < 2: return ''
            tokens = tokens[:-1]
        if len(tokens) == 1: return ''
        return tokens[-1].split('#')[-1].split('/')[-1].strip()

    def __handle_unicode(self, entity):
        numeric_code = int(entity[2:-1])
        if numeric_code >= 0x10000: return ''
        return unichr(numeric_code)

#-------------------------------------------------------------------------------

class OutputSplitter:
    def __init__(self, compress, max_file_size, path_name):
        self.__dir_index = 0
        self.__file_index = -1
        self.__cur_file_size = 0
        self.__compress = compress
        self.__max_file_size = max_file_size
        self.__path_name = path_name
        self.__out_file = self.__open_next_file()

    def write(self, text):
        text_len = len(text)
        if self.__cur_file_size + text_len / 2 > self.__max_file_size:
            self.__close_cur_file()
            self.__out_file = self.__open_next_file()
            self.__cur_file_size = 0
        self.__out_file.write(text)
        self.__cur_file_size += text_len

    def close(self):
        self.__close_cur_file()

    def __open_next_file(self):
        self.__file_index += 1
        if self.__file_index == 100:
            self.__dir_index += 1
            self.__file_index = 0
        dir_name = self.__get_dir_name()
        if not os.path.isdir(dir_name):
            os.makedirs(dir_name)
        file_name = os.path.join(dir_name, self.__get_file_name())
        if not self.__compress:
            return open(file_name, 'w')
        else:
            return bz2.BZ2File('%s.bz2' % file_name, 'w')

    def __close_cur_file(self):
        self.__out_file.close()

    def __get_dir_name(self):
        char1 = self.__dir_index % 26
        char2 = self.__dir_index / 26 % 26
        return os.path.join(self.__path_name, '%c%c' % (65 + char2, 65 + char1))
    def __get_file_name(self):
        return 'wiki%02d' % self.__file_index

### CORE ###############################################################

def process_data(input_file, wiki_extractor, output_splitter):
    page = []
    for line in input_file:
        line = line.decode('utf-8').strip()
        if line == '<page>':
            page = []
        elif line == '</page>':
            process_page(page, wiki_extractor, output_splitter)
        else:
            page.append(line)

#----------------------------------------------------------------------------

def process_page(page, wiki_extractor, output_splitter):
    wiki_document = extract_document(page)
    if not wiki_document: return

    wiki_document = wiki_extractor.extract(wiki_document)
    if not wiki_document: return

    output_splitter.write(wiki_document.__str__().encode('utf-8'))

#----------------------------------------------------------------------------

def extract_document(page):
    wiki_document = WikiDocument()
    for line in page:
        if not line: continue

        # Identificatore della pagina (nodo XML)
        if not wiki_document.id and line.startswith('<id>') and line.endswith('</id>'):
            wiki_document.id = int(line[4:-5])
            continue
        # Titolo della pagina (nodo XML)
        elif not wiki_document.url and line.startswith('<title>') and
line.endswith('</title>'):
            title = line[7:-8].replace('&amp;', '&')
            if ':' in title: return None
            quoted_title = urllib.quote(title.replace(' ', '_').encode('utf-8'))
            wiki_document.url = 'http://sv.wikipedia.org/wiki/%s' % quoted_title
            wiki_document.text = '++%s++' % title
            continue
        # Inizio del testo della pagina (nodo XML)
        elif line.startswith('<text'):
            if line.endswith('</text>'): return None
            line = line[27:]
            if not line: continue
        # Fine del testo della pagina (nodo XML)
        elif line.endswith('</text>'):
            line = line[:-7]
            if not line: continue
        # Informazione superflua (nodo XML)
        elif line[0] == '<':
            continue
        # Titolo di paragafo (testo della pagina)
     #   elif line[0] == '=':
      #      line = '==%s==' % line.strip('= ')

        wiki_document.text += '\n%s' % line

    return wiki_document

### USER INTERFACE ######################################################

def show_help():
    print >> sys.stdout, __doc__,

def show_usage(output_file, script_name):
    print >> output_file, 'Usage: %s [options]' % script_name

def show_suggestion(output_file, script_name):
    print >> output_file, 'Try \'%s --help\' for more information.' % script_name

def show_size_error(script_name, file_size):
    print >> sys.stderr, '%s: %s: Insufficient or invalid number of bytes' % (script_name,
file_size)

def show_file_error(script_name, file_name):
    print >> sys.stderr, '%s: %s: No such file or directory' % (script_name, file_name)

def main():
    script_name = os.path.basename(sys.argv[0])

    try:
        long_opts = ['help', 'usage', 'compress', 'bytes=', 'output=']
        opts, args = getopt.gnu_getopt(sys.argv[1:], 'cb:o:', long_opts)
    except getopt.GetoptError:
        show_usage(sys.stderr, script_name)
        show_suggestion(sys.stderr, script_name)
        sys.exit(1)

    compress = False
    file_size = 500 * 1024
    output_dir_name = '.'

    for opt, arg in opts:
        if opt == '--help':
            show_help()
            sys.exit()
        elif opt == '--usage':
            show_usage(sys.stdout, script_name)
            sys.exit()
        elif opt in ('-c', '--compress'):
            compress = True
        elif opt in ('-b', '--bytes'):
            try:
                if arg[-1] in 'kK':
                    file_size = int(arg[:-1]) * 1024
                elif arg[-1] in 'mM':
                    file_size = int(arg[:-1]) * 1024 * 1024
                else:
                    file_size = int(arg)
                if file_size < 200 * 1024: raise ValueError()
            except ValueError:
                show_size_error(script_name, arg)
                sys.exit(2)
        elif opt in ('-o', '--output'):
            if os.path.isdir(arg):
                output_dir_name = arg
            else:
                show_file_error(script_name, arg)
                sys.exit(3)

    if len(args) > 0:
        show_usage(sys.stderr, script_name)
        show_suggestion(sys.stderr, script_name)
        sys.exit(4)

    wiki_extractor = WikiExtractor()
    output_splitter = OutputSplitter(compress, file_size, output_dir_name)
    process_data(sys.stdin, wiki_extractor, output_splitter)

    output_splitter.close()

if __name__ == '__main__':
    main()
```

Appendix D *Java Code for Building the Wikipedia Corpus*

```
import java.io.*;
import java.text.*;
import java.util.*;
public class main {
        public static void main(String[] args){
                FileInputStream fstream;
                InputStreamReader input;
                BufferedReader br;
                FileWriter fstreamout;
                Locale currentLocale = new Locale ("sv","SE");
                BreakIterator sentenceIterator =
                BreakIterator.getSentenceInstance(currentLocale);
                String text = "";
                try {
                        fstream = new
                        FileInputStream("E:\\exjobb\\wikiextractor\\extracted1\\AA\\wiki00");
                        input = new InputStreamReader(fstream, "UTF-8");
                        br = new BufferedReader(input);
                        fstreamout = new FileWriter("e:\\exjobb\\wikiout.txt");
                        BufferedWriter out = new BufferedWriter(fstreamout);
                        String line = "";
                        while (( line = br.readLine()) != null){
                                if(line.contains("<doc id=")){
                                        markBoundaries(text, sentenceIterator,out);
                                        out.write(line + "\n");
                                        String title = br.readLine();
                                        out.write("<Title>" + title.substring(0, title.length()-1)
                                        + "</Title>" + "\n");
                                        text = "";
                                }else if(line.contains("======")){
                                        markBoundaries(text, sentenceIterator,out);
                                        line = line.replaceAll("======(.*?)======",
                                        "<H5>$1</H5>");
                                        out.write(line+ "\n");
                                        text = "";
                                }else if(line.contains("=====")){
                                        markBoundaries(text, sentenceIterator,out);
                                        line = line.replaceAll("=====(.*?)=====", "<H4>$1</H4>");
                                        out.write(line+ "\n");
                                        text = "";
                                }else if(line.contains("====")){
                                        markBoundaries(text, sentenceIterator,out);
                                        line = line.replaceAll("====(.*?)====", "<H3>$1</H3>");
                                        out.write(line + "\n");
                                        text = "";
                                }else if(line.contains("===")){
                                        markBoundaries(text, sentenceIterator,out);
                                        line = line.replaceAll("===(.*?)===", "<H2>$1</H2>");
                                        out.write(line+ "\n");
                                        text = "";
                                }else if(line.contains("==")){
                                        markBoundaries(text, sentenceIterator,out);
                                        line = line.replaceAll("==(.*?)==", "<H1>$1</H1>");
                                        out.write(line + "\n");
                                        text = "";
                                }else if(line.contains("</doc>")){
                                        markBoundaries(text, sentenceIterator,out);
                                        out.write(line + "\n");
                                        text = "";
                                }else{
                                        text += line+ "\n";
                                }
                        }
                        out.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
static void markBoundaries(String target, BreakIterator iterator,BufferedWriter out) {
                iterator.setText(target);
                int boundary2 = iterator.first();
                int boundary1 = 0;
                while (boundary2 != BreakIterator.DONE) {
                        try {
                                if(!(target.substring(boundary1,boundary2) == "\n") &&
                                boundary1!=boundary2){
                                        String sentence = target.substring(boundary1,boundary2);
                                        if(sentence.endsWith("\n")){
                                                sentence = sentence.substring(0,
                                                sentence.length()-1);
                                        }
                                        out.write("<S> "+sentence+" </S>" + "\n");
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                        boundary1 = boundary2;
                        boundary2 = iterator.next();
                }

        }
}
```

Appendix E *Java Code for Building the Crawled Corpus*

Class main:

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.StringReader;
import java.text.BreakIterator;
import java.util.ArrayList;
import java.util.Locale;
import javax.swing.text.html.HTMLEditorKit;
import org.knallgrau.utils.textcat.TextCategorizer;
public class main {
        public static String currentEncoding = "iso-8859-1";
        public static void main(String[] args){
                ParserGetter kit = new ParserGetter();
                HTMLEditorKit.Parser parser = kit.getParser();
                HTMLEditorKit.ParserCallback callback = null;
                PrintStream swe = null;
                PrintStream junk = null;
                int count = 0;
                int id = 0;
                ArrayList<String> oldWords = new ArrayList<String>();
                oldWords.add(" hv");
                oldWords.add(" åro ");
                oldWords.add(" gingo ");
                oldWords.add(" gåfvor ");
                oldWords.add(" gåfva ");
                oldWords.add(" hafva ");
                oldWords.add(" fingo ");
                oldWords.add(" voro ");
                oldWords.add(" blevo ");
                oldWords.add(" bedrevo ");
                oldWords.add(" däröfver ");
                oldWords.add(" ifver ");
                oldWords.add(" blefvo ");
                oldWords.add(" blifva ");
                oldWords.add(" lofva ");
                oldWords.add(" blifver ");
                oldWords.add(" lefver ");
                oldWords.add(" afrätta ");
                oldWords.add(" kufva ");
                oldWords.add(" afskära ");
                oldWords.add(" afrätta ");
                oldWords.add(" afsky ");
                try {
                        FileOutputStream fopsswe = new
FileOutputStream("c:\\outputSWE.txt");
                        FileOutputStream fopjunk = new FileOutputStream("c:\\junk.txt");
                        swe = new PrintStream(fopsswe, true,"iso-8859-1");
                        junk = new PrintStream(fopjunk,true,"iso-8859-1");
                        } catch (Exception e1) {
                        e1.printStackTrace();
                }
                try{
                        FileInputStream freader = new FileInputStream("c:\\dumptotal");
                        InputStreamReader instrR = new InputStreamReader(freader,"iso-8859-
1");
                        BufferedReader bfr = new BufferedReader(instrR);
                        String[] result = new String[3];
                        while(result != null){
                                swe.flush();
                                junk.flush();
                                result[0]="";
                                result[1]="";
                                result[2]="";
                                result = getContent(bfr,freader);
                                StringReader sr = new StringReader(result[1]);
                                if(!result[2].equals("") ){
                                        int a = writeSentence(result[2],swe,junk, result[0],
oldWords,id);
                                        count +=a;
                                        if(a>0){
                                                id++;
                                                try{
                                                        callback = new TagStripper(swe);
                                                        parser.parse(sr,callback,true);
                                                }catch(StackOverflowError e){
                                                        System.out.println("Could not parse
page");
                                                }
                                                swe.println("</doc>");
                                        }else{
                                                try{
                                                        callback = new TagStripper(junk);
                                                        parser.parse(sr,callback,true);
                                                }catch(StackOverflowError e){
                                                        System.out.println("Could not parse
page");
                                                }
                                                junk.println("</doc>");
                                        }
                                }
                        }
                }catch(Exception e){
                        System.err.println(e);
                }
                System.out.println("Number Of Sentences: " +count);
        }
        private static String[] getContent(BufferedReader reader, FileInputStream
freader){
                String temp="";
                String result[]= new String[3];
                result[0]="";
                result[1]="";
                result[2]="";
                if(!currentEncoding.equals("iso-8859-1")){
                        currentEncoding = "iso-8859-1";
                }
                try {
                        temp = reader.readLine();
                } catch (IOException e1) {
                        e1.printStackTrace();
                }
                while(true){
                        try {
                                if(temp == null){
                                        return null;
                                }
                                if(temp.contains("Recno::")){
                                        return result;
                                }
                                if(temp.contains("ParseText::")){
                                        result[2] = reader.readLine();
                                }
                                if(temp.contains("metadata:")&&temp.contains("charset=")){
                                        String encoding =
temp.replaceAll(".*charset=(\\S*).*", "$1");
                                        encoding = encoding.toLowerCase();
                                        if(!currentEncoding.equals(encoding)){
                                                currentEncoding = encoding;
                                        }
                                }
                                if(temp.contains("URL::")){
                                        result[0]= temp;
                                }else{
                                        result[1] = result[1] + "\n" + temp;
                                }
                                temp = reader.readLine();
                                if(currentEncoding.equals("utf-8")){
                                        byte[] buf = temp.getBytes();
                                        temp = new String(buf, "UTF-8");

                                }

                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        }
        static int writeSentence(String text, PrintStream out,PrintStream junk, String
url, ArrayList<String> oldWords, int id) {
                double nbrOfSentences=0;
                int count=0;
                String result = "";
                TextCategorizer textCat = new TextCategorizer();
                String sentence = "";
                ArrayList<String> sentences = new ArrayList<String>();
                try{
                        Locale currentLocale = new Locale ("sv","SE");
                        BreakIterator sentenceIterator =
BreakIterator.getSentenceInstance(currentLocale);
                        sentenceIterator.setText(text);
                        int boundary2 = sentenceIterator.first();
                        int boundary1 = 0;
                        boolean sentenceOk = false;
                        while (boundary2 != BreakIterator.DONE) {
                                if(text.substring(boundary1,boundary2) == "\n" ||
boundary1==boundary2){
                                }else{
                                        sentence = text.substring(boundary1,boundary2);
                                        while(sentence.startsWith(" ")||
sentence.startsWith(" ")){
                                                sentence = sentence.substring(1);
                                        }
                                        while(sentence.endsWith(" ")||
sentence.endsWith(" ")){
                                                sentence = sentence.substring(0,
sentence.length()-1);
                                        }
                                        while(sentence.endsWith("\n")||
sentence.endsWith("\r")){
                                                sentence = sentence.substring(0,
sentence.length()-1);
                                        }
                                        sentenceOk = sentenceIsOk(sentence, oldWords);

                                        if(sentenceOk){
                                                nbrOfSentences++;
                                                sentences.add(sentence);
                                                result += sentence + " ";
                                        }else{
                                                junk.println("<S> "+sentence+" </S>");
                                        }
                                }
                                boundary1 = boundary2;
                                boundary2 = sentenceIterator.next();
                        }
                }catch(Exception e){
                        System.out.println(" SENTENCE CAST EXCEPTION   "+ sentence);
                }
                if((textCat.categorize(result).equals("swedish") &&
wordsOk(result,oldWords))){
                        url = url.replace("URL:: ","");
                        out.println("<doc id =\"" + id +"\" url = \"" + url + "\">");
                        for(int i=0;i<sentences.size();i++){
                                String s = sentences.get(i);
                                if(s.charAt(s.length()-1)=='.' && (!s.endsWith(" .") ||
!s.endsWith(" ."))){
                                        s = s.substring(0, s.length()-1);
                                        s = s + " .";
                                }
                                if(s.charAt(s.length()-1)=='!' && (!s.endsWith(" !") ||
!s.endsWith(" !"))){
                                        s = s.substring(0, s.length()-1);
                                        s = s + " !";
                                }
                                if(s.charAt(s.length()-1)=='?' && (!s.endsWith(" ?") ||
!s.endsWith(" ?"))){
                                        s = s.substring(0, s.length()-1);
                                        s = s + " ?";
                                }
                                out.println("<S> "+s+" </S>");
```

```
                            count++;
                    }
                    return count;


            }else{
                    url = url.replace("URL:: ","");
                    junk.println("<doc " + "\" url = \"" + url + "\">");
                    for(int i=0;i<sentences.size();i++){
                            junk.println("<S>"+sentences.get(i)+"</S>");
                    }
                    junk.println("</doc>");
            }
            return 0;
    }
    private static boolean sentenceIsOk(String sentence, ArrayList<String> oldWords) {

            if(sentence.length() == 0){
                    return false;
            }
            if(sentence.contains("  ")|| sentence.contains("  ")){
                    System.out.println("2 spaces " + sentence);
                    return false;
            }
            if(!sentence.contains(" ")|| !sentence.contains(" ")){
                    System.out.println("Only 1 word " + sentence);
                    return false;
            }
            if(sentence.contains(" . ")){
                    System.out.println("Hanging dot " + sentence);
                    return false;
            }
            if(sentence.contains(" . ")){
                    System.out.println("Hanging dot " + sentence);
                    return false;
            }
            if(sentence.contains(".se") || sentence.contains(".com") ||
sentence.contains(".nu")|| sentence.contains(".org")|| sentence.contains(".net")||
sentence.contains("www.")){
                    return false;
            }
            if(sentence.contains("--")){
                    System.out.println("contains -- " + sentence);
                    return false;
            }
            for(int i = 0; i<sentence.length(); i++){
                    if(!Character.isLetterOrDigit(sentence.charAt(i)) &&
!(sentence.charAt(i)=='.') && !(sentence.charAt(i)== ',')&& !(sentence.charAt(i)== '?')
&& !(sentence.charAt(i)== '!') && !(sentence.charAt(i) == '&') && !(sentence.charAt(i) ==
' ') && !(sentence.charAt(i) == ' ') && !(sentence.charAt(i) == '(') &&
!(sentence.charAt(i) == ')') && !(sentence.charAt(i) == '-') &&
!(sentence.charAt(i)=='"')&& !(sentence.charAt(i)==':')&& !(sentence.charAt(i)==';')&&
!(sentence.charAt(i)=='/')&& !(sentence.charAt(i)=='\'')){
                            return false;
                    }
            }
            return true;
    }
    private static boolean wordsOk(String result, ArrayList<String> oldWords){
            for(int i = 0; i<oldWords.size(); i++){
                    if(result.contains(oldWords.get(i))){
                            return false;
                    }
            }
            return true;
    }
}
```

Class ParserGetter

```
import javax.swing.text.html.*;
public class ParserGetter extends HTMLEditorKit {

    public HTMLEditorKit.Parser getParser() {
        return super.getParser();
    }
}
```

Class TagStripper

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import java.io.Writer;
import org.knallgrau.utils.textcat.TextCategorizer;

import javax.swing.text.BadLocationException;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.html.*;
import javax.swing.text.html.HTML.Tag;


public class TagStripper extends HTMLEditorKit.ParserCallback {
        private boolean inBody = false;
        private boolean inP = false;
        private boolean H1 = false;
        private boolean H2 = false;
        private boolean H3 = false;
        private boolean H4 = false;
        private boolean H5 = false;
        private boolean H6 = false;
        private boolean LI = false;
        private String output = "";
        private String taggedOutput = "";
        private TextCategorizer textCat = new TextCategorizer();

        private PrintStream out;

        public TagStripper(PrintStream out) {

                this.out=out;

        }

        @Override
        public void handleStartTag(Tag tag, MutableAttributeSet a,int position) {

                if(tag == HTML.Tag.BODY){
                        inBody = true;
                }
                if(tag == HTML.Tag.H1){
                        H1 = true;
                }
                if(tag == HTML.Tag.H2){
                        H2 = true;
                }
                if(tag == HTML.Tag.H3){
                        H3 = true;
                }
                if(tag == HTML.Tag.H4){
                        H4 = true;
                }
                if(tag == HTML.Tag.H5){
                        H5 = true;
                }
                if(tag == HTML.Tag.H6){
                        H6 = true;
                }
        }

        public void handleEndTag(Tag tag,int position){
```

```
        if(tag == HTML.Tag.BODY){
                inBody = false;
        }
        if(tag == HTML.Tag.H1){
                H1 = false;
        }
        if(tag == HTML.Tag.H2){
                H2 = false;
        }
        if(tag == HTML.Tag.H3){
                H3 = false;
        }
        if(tag == HTML.Tag.H4){
                H4 = false;
        }
        if(tag == HTML.Tag.H5){
                H5 = false;
        }
        if(tag == HTML.Tag.H6){
                H6 = false;
        }
        if (tag == HTML.Tag.HTML)
                try {
                        this.flush();
                } catch (BadLocationException e) {
                        System.out.println("test");
                        e.printStackTrace();
                }
}
@Override
public void flush() throws BadLocationException {
        if(output.equals("")||output.equals("\n")||output.equals("\n "))
                return;
        byte[] buf = taggedOutput.getBytes();
        out.println(taggedOutput);
        output = "";
        taggedOutput="";
        super.flush();
}

@Override
public void handleText(char[] text, int position) {

        if(text.equals(" "))
                return;
        if(H1){
                String heading = String.valueOf(text);
                while(heading.startsWith(" ")|| heading.startsWith(" ")){
                        heading = heading.substring(1);
                }
                while(heading.endsWith(" ")|| heading.endsWith(" ")){
                        heading = heading.substring(0, heading.length()-1);
                }
                if(headingIsOk(heading)){
                        taggedOutput =taggedOutput + "\n" +"<H1>"+heading+"</H1>";
                        output = output + "\n" + heading;
                }
        }
        if(H2){
                String heading = String.valueOf(text);
                while(heading.startsWith(" ")|| heading.startsWith(" ")){
                        heading = heading.substring(1);
                }
                while(heading.endsWith(" ")|| heading.endsWith(" ")){
                        heading = heading.substring(0, heading.length()-1);
                }
                if(headingIsOk(heading)){
                        taggedOutput =taggedOutput + "\n" +"<H2>"+heading+"</H2>";
                        output = output + "\n" + heading;
                }
        }
        if(H3){
                String heading = String.valueOf(text);
                while(heading.startsWith(" ")|| heading.startsWith(" ")){
                        heading = heading.substring(1);
                }
                while(heading.endsWith(" ")|| heading.endsWith(" ")){
                        heading = heading.substring(0, heading.length()-1);
                    }
                    if(headingIsOk(heading)){
                        taggedOutput =taggedOutput + "\n" +"<H3>"+heading+"</H3>";
                        output = output + "\n" + heading;
                    }
                }
                if(H4){
                    String heading = String.valueOf(text);
                    while(heading.startsWith(" ")|| heading.startsWith(" ")){
                        heading = heading.substring(1);
                    }
                    while(heading.endsWith(" ")|| heading.endsWith(" ")){
                        heading = heading.substring(0, heading.length()-1);
                    }
                    if(headingIsOk(heading)){
                        taggedOutput =taggedOutput + "\n" +"<H4>"+heading+"</H4>";
                        output = output + "\n" + heading;
                    }
                }
                if(H5){
                    String heading = String.valueOf(text);
                    while(heading.startsWith(" ")|| heading.startsWith(" ")){
                        heading = heading.substring(1);
                    }
                    while(heading.endsWith(" ")|| heading.endsWith(" ")){
                        heading = heading.substring(0, heading.length()-1);
                    }
                    if(headingIsOk(heading)){
                        taggedOutput =taggedOutput + "\n" +"<H5>"+heading+"</H5>";
                        output = output + "\n" + heading;
                    }
                }
                if(H6){
                    String heading = String.valueOf(text);
                    while(heading.startsWith(" ")|| heading.startsWith(" ")){
                        heading = heading.substring(1);
                    }
                    while(heading.endsWith(" ")|| heading.endsWith(" ")){
                        heading = heading.substring(0, heading.length()-1);
                    }
                    if(headingIsOk(heading)){
                        taggedOutput =taggedOutput + "\n" +"<H6>"+heading+"</H6>";
                        output = output + "\n" + heading;
                    }
                }
        }

        private boolean headingIsOk(String heading) {
                if(heading.length() == 0){
                    return false;
                }
                for(int i = 0; i<heading.length(); i++){
                    if(!Character.isLetterOrDigit(heading.charAt(i)) &&
!(heading.charAt(i)=='.') && !(heading.charAt(i)== ',')&& !(heading.charAt(i)== '?') &&
!(heading.charAt(i)== '!') && !(heading.charAt(i) == '&') && !(heading.charAt(i) == ' ')
&& !(heading.charAt(i) == ' ') && !(heading.charAt(i) == '(') && !(heading.charAt(i) ==
')') && !(heading.charAt(i) == '-')){
                        return false;
                    }
                }
                return true;
        }
}
```

Appendix F *Example of a corpus entry from Wikipedia*

```
<doc id="1" url="http://sv.wikipedia.org/wiki/Amager">
<Title>Amager</Title>
<S>Amager är en dansk ö i Öresund med 160 000 invånare.</S>
<S>Delar av Köpenhamn ligger på ön, och övriga delar upptas av Tårnby kommun och Dragør
kommun.</S>
<S>Amager är delvis en konstgjord ö, delvis en naturlig sådan. </S>
<S>Ön är mycket låg och vissa delar ligger under havsytan, framförallt Vestamager. </S>
<S>Ön började uppodlas i större skala under 1500-talets första hälft då kung Kristian II
bjöd in nederländska bönder till att odla upp ön för kronans räkning. </S>
<S>Flertalet slog sig ned i fiskeläget Dragör där många än idag har nederländska namn som
Neels och Tønnes.</S>
<S>Den stora utvecklingsfasen började på 1800-talet då flertalet fabriker, bostadsområden
och militär verksamhet utvecklade sig på ön. </S>
<S>Ön, som är förbunden med övriga Köpenhamn genom flera broar, järnväg och Metro, har
alltid varit ett arbetarklassfäste med undantag för Dragör och vissa andra områden. </S>
<S>Ön har aldrig ansetts vara riktigt "fin" i många köpenhamnsbors ögon, men
inställningen håller på att ändras i takt med stigande huspriser.</S>
<S>Under 1900-talet började militären att utveckla stora övningsområden på ön vilket
resulterade i Vestamager som idag innefattar ett stort naturområde ett par kilometer från
Köpenhamns centrum. </S>
<S>Under senare delen av 1900-talet har flygplatsen Kastrup som ligger på ön expanderat
kraftigt och i samband med byggandet av Öresundsförbindelsen började även ett stort
område byggas som kallas för Örestad. </S>
<S>Ön återfick i samband med bron även en järnvägsförbindelse vilket tidigare inte
funnits sedan 1960-talet. </S>
<S>En motorväg byggdes samtidigt som går rakt igenom flera tidigare bostadsområden. </S>
<S>Köpenhamns metro håller på att byggas ut till Kastrup och flera andra stora
byggprojekt är på gång. </S>
<S>Många "amagerkanare" [ama'rkänare] ser dock denna utveckling med stor skepsis eftersom
de menar att denna gentrifiering håller på att förstöra deras ö med nyrika och
"trendsättare" som flyttar in där arbetarklassen och fabrikerna tidigare huserade.</S>
<S>Amager är den tätast befolkade ön i Danmark. </S>
<S>Den näst mest tätbefolkade är Thurø.</S>
</doc>
```

Appendix G *Lua Script to Create 3-gram Statistics*

```
#!/usr/bin/env lua


lines = 0
file = io.open("ngram", "w")

-- Count must give the value 0 for non existing key
count = {}
mt = {}
mt.__index = function() return 0 end
setmetatable(count, mt)

-- Count
for line in io.lines() do
        lines = lines+1
        word1 = ""
        word2 = ""
        word3 = ""
        word4 = ""
        word5 = ""
        for word in string.gfind(line, "[^%s]+") do
                word1 = word2
                word2 = word3
                word3 = word
                if #word1 > 0 then
                        trigram = word1 .. " " .. word2 .. “ “ .. word3
                        count[trigram] = count[trigram] + 1
                end
        end
end
-- print
for k, c in pairs(count) do
        if(c >= 10) then
                file:write(k .. "\t" .. c .. "\n")
        end
end
```