StupidOS: A high-precision approach to boilerplate removal

Stefan Evert [*]
University of Osnabrück

Abstract

StupidOS is a naive approach to boilerplate removal from English Web pages, based on n-gram models trained on the FIASCO gold standard. Despite its simplicity, the algorithm achieves a significant improvement over the baseline and trades off recall for substantially higher precision. Even the prototype implementation of StupidOS in pure Perl is lightweight and fast enough to process billions of words.

Keywords : CLEANEVAL, Web as corpus, boilerplate, text cleanup, n-gram models.

1. Introduction

The World Wide Web is a fantastic, almost inexhaustible, and very convenient source of authentic language data. Automatically compiled Web corpora have become increasingly popular in recent years and have been used for many purposes (Kilgarriff & Grefenstette 2003; Baroni & Bernardini 2006). In particular, Web corpora offer the NLP community an opportunity to train statistical models on, and mine information from, much larger amounts of text than was previously possible. The WaCky initiative [1] has collected 1–2 billion words each of English, German and Italian with relative ease, and many more large Web corpora are expected to become available within the next few years.

While statistical models are often applied to large and messy corpora in the hope that the errors will somehow "cancel out", Web pages are even more messy than other text sources. Therefore, thorough preprocessing and "cleanup" of the pages is crucial if frequency data obtained from Web corpora are to have any degree of dependability.

First of all, many of the HTML pages downloaded by a Web crawler do not contain "ordinary" text, written by human (and preferably native) speakers and intended for human readers, but rather advertising (with many images, animations and hidden keyword lists), Web spam (artificial pages constructed to influence search engine rankings), automatically generated pages (e.g. from eBay, Amazon, etc.), Web directories and search results (often fake ones luring visitors to click on advertisements), as well as many other types of undesirable material. Such pages need to be filtered out from the Web crawl. For

[*] Institute of Cognitive Science, University of Osnabrück, stefan.evert@uos.de

[1] `http://wacky.sslmit.unibo.it/`

the remaining pages, character encoding has to be determined correctly (which can be fairly difficult for languages such as Russian, Chinese and Japanese, where international standards were adopted at a much later time than for Western languages), and pages that are not written in the language of interest have to be removed.

The second major problem of Web corpora is the abundance of duplicate pages on the Web. In the simplest case, these are perfect duplicates, where the same page is available from different URLs.[2] It is much more difficult to recognise near-duplicate pages, which have (largely) the same original content, but different layout and design (headers, footers, navigation bars, etc.). A substantial amount of duplication can also be found at a finer granularity, where the ease of cut & paste encourages reuse of text blocks, stories, etc. found somewhere on the Web. Bulletin boards and Weblogs (often with long lists of reader comments) also tend to contain many copies of the same text segments, typically in the form of quotations. Since such duplicates inflate frequency counts in a way that statistical methods cannot account for, it is essential to detect them with high accuracy and discard all copies of each text but one.

Finally, even when "good" Web pages have been identified and converted to Unicode text, most of them still contain a substantial amount of unwanted material. This has become known as *boilerplate* – canned or automatically generated text blocks including navigation bars, page headers, link lists, disclaimers and copyright statements, as well as the ubiquitous advertisements. Boilerplate has a similar effect as page duplicates, in that it drastically inflates frequency counts for certain terms and expressions (such as *click here*, *Contents* or *Vi@gr@*). Therefore, reliable *boilerplate removal* is indispensable to make Web corpora useful for linguistic and statistical analysis.

Many of the problems of Web corpora listed above are shared by search engines, information mining, and other popular fields of research. Therefore, sophisticated methods have been developed for the recognition of character encoding and language, the identification of Web spam, as well as duplicate detection. These tools can readily be applied by the compilers of a Web corpus. Search engines and information mining applications are usually not concerned about boilerplate, though, since search results can relatively easily be filtered or ranked to reduce the influence of unwanted text material. On the other hand, boilerplate is a serious problem for Web corpora, where it injects substantial amounts of non-authentic and highly duplicative text. On average, 20% or more of the tetxual content of a Web page consists of boilerplate text (as can be estimated from the development set of manually cleaned pages released by the CLEANEVAL organisers). Nonetheless, the task of boilerplate removal has largely been trivialised or ignored in the Web as corpus community prior to the CLEANEVAL contest.

This paper introduces a naive algorithm for boilerplate removal, aptly named *StupidOS*, which makes use of the English gold standard produced by the FIASCO team participating in CLEANEVAL (Bauer *et al.* 2007). StupidOS trains simple character-level n-gram

[2] For instance, `www.collocations.de`, `www.collocations.de/index.html`, `www.schtepf.de/collocations`, `www.schtepf.de/collocations/` and `www.schtepf.de/collocations/index.html` all point to the same Web page.

models on the gold standard in order to discriminate between boilerplate (called *dirty* text) and "good" (or *clean*) text. The system has been evaluated both on the FIASCO gold standard and the official CLEANEVAL development set. In both cases, its performance is significantly better than the baseline and constitutes a valuable improvement over straightforward text dumps. Compared to such text dumps, StupidOS leads to an overall improvement in quality, but also trades off recall for substantially higher precision. Given the almost unlimited amount of text that can be downloaded from the Web, this high-precision approach to boilerplate removal seems warranted.

However, the StupidOS algorithm is far from perfect and more research will be needed to obtain at least a subset of pages where all boilerplate has reliably been deleted (perhaps at the expense of some clean text).

2. The FIASCO gold standard

The development data sets of manually cleaned Web pages released by the CLEANEVAL organisers, containing 58 pages for English and 51 for Chinese, do not provide sufficient training data for machine-learning approaches to boilerplate removal. In February 2007, at a time when not even these development sets were available, the FIASCO team (Bauer *et al.* 2007) decided to develop their own gold standard of English Web pages for participation in the CLEANEVAL contest. The gold standard consists of 158 Web pages from several topical domains (including "Norway", "computer hardware", "photography" and "Japanese manga and anime"), amounting to a total of more than 300,000 words and almost 2 million characters (after manual cleanup). The same procedure and guidelines were used as for the CLEANEVAL development set. Text dumps of the Web pages were cleaned up manually, then text segments were identified and classified as `<p>` (normal paragraph), `<h>` (heading) or `<l>` (list item). Each page was independently cleaned by two annotators, and conflicts were resolved by a third person. With some extentions and refinements to the CLEANEVAL guidelines, relatively high inter-annotator agreement (above 90%) was achieved.

The FIASCO gold standard suffers from the same mistake as the CLEANEVAL development set: both procedures started from raw text dumps, which were edited manually to remove boilerplate and insert segment markers. In this way, a reference set of cleaned text files was obtained, but information was lost about which parts of the original Web page were boilerplate and which contained clean text. As the FIASCO team later found out, it is a non-trivial task to reconstruct an alignment between the HTML tree (or even the raw text dumps) and the manually cleaned text, and no satisfactory alignment quality has been achieved so far (Bauer *et al.* 2007 : Sec. 4). As a consequence, the FIASCO gold standard also fails to provide a corpus of boilerplate (consisting of the dirty text removed from the gold standard files), which could be used to train a supervised classifier or statistical language model to recognise boilerplate text segments. All that is available is a corpus of clean text, plus the raw unedited text dumps. This lack of dirty training data was one of the main obstacles that had to be overcome in the implementation of the StupidOS system.

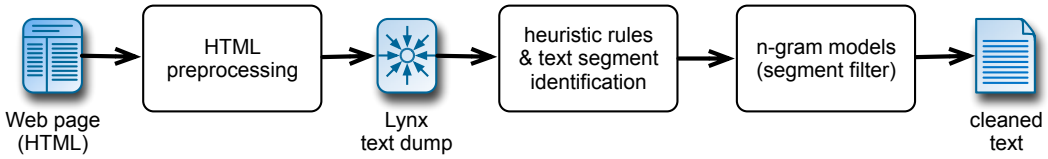

Figure 1. Architecture of the StupidOS system

3. The StupidOS algorithm

The research question that motivated the development of StupidOS was to find out whether accurate boilerplate removal is possible working exclusively on plain text dumps, without access to the structure of the original HTML page. This is particularly interesting because most previous approaches, such as the *PotaModule*, [3] rely to a large extent on heuristic measures of HTML tag density (in addition to sets of hand-selected keywords). Making use of the FIASCO gold standard as training data, a simple and straightforward (some might say *naive*) language modelling approach was implemented to distinguish between clean and dirty text.

3.1. System architecture

The StupidOS system is a pipeline of four components, as illustrated in Fig. 1.

The *first* component performs some normalisation on the original HTML code. Regular expressions are used to delete images, comments and inline Javascript. In addition, special markers are inserted for headings and list items, and `<br/>` tags are converted to regular paragraph breaks.

In the *second* step, the pre-processed HTML page is converted to plain text using the Lynx browser. [4] StupidOS relies on the built-in character encoding recognition of the browser, and obtains a text file in UTF-8 format.

In a *third* post-processing step, invalid characters are removed from the text, and some easily recognisable types of boilerplate (e.g. navigation bars or footer blocks where multiple fields are separated by vertical bars "`|`") are deleted with regular expressions. Itemised and enumerated lists in the Lynx output are detected, as well as paragraph breaks indicated by empty lines. In this way, the text is broken down into segments, which are classified as `<p>`, `<h>` or `<l>` (partly relying on the markers inserted by the first component).

The *fourth* and core component of the StupidOS algorithm are n-gram language models for clean and dirty text. These models are applied to each text segment in turn. If the dirty model calculates a higher probability than the clean model, the text segment is considered to be boilerplate and deleted. Otherwise it is included in the final output.

A research prototype of the StupidOS system has been implemented in pure Perl (including the n-gram models) and is therefore platform-independent.

[3] `http://sslmitdev-online.sslmit.unibo.it/wac/post_processing.php`
[4] `http://lynx.browser.org/`

3.2. The language models

In order to distinguish between boilerplate and clean text, the StupidOS system uses simple character-level n-gram language models (Manning & Schütze 1999), based on conditional probabilities $\Pr(c_n \,|\, c_1 \ldots c_{n-1})$, where $c_i \in \Sigma$ are characters in the input alphabet (here restricted to the ASCII range). In order to compensate for the small, uniform training corpus and ensure better generalisation, geometric interpolation was applied to estimate n-gram probabilities for unseen text:

$$\Pr_*(c_n \,|\, c_1 \ldots c_{n-1}) = \frac{1-q}{1-q^n} \cdot \Big(\Pr(c_n \,|\, c_1 \ldots c_{n-1}) + q \cdot \Pr(c_n \,|\, c_2 \ldots c_{n-1}) + \cdots + q^{n-1} \cdot \Pr(c_n) \Big)$$

In addition, add-one smoothing was applied to the unigram probabilities $\Pr(c_n)$ in order to account for unseen or infrequent characters in the training data. Optimisation of the interpolation parameter $q \in (0,1)$ is described in Sec. 3.3. Note that $q \approx 0$ corresponds to an unsmoothed n-gram model without interpolation, whereas $q \approx 1$ gives equal weight to all history sizes. Since English text consists almost exclusively of ASCII characters (except for accented letters in a small number of loan words, as well as foreign words and proper names), non-ASCII characters were replaced by tilde symbols (˜) to reduce data sparseness problems.

Model training consisted in simple maximum-likelihood estimation of the required conditional probabilities from training data. This procedure was straightforward for the clean language model trained on the gold standard corpus of cleaned Web pages. Unfortunately, a corresponding training corpus for the dirty model was not available, as explained in Sec. 2. In order to overcome this problem, a *differential training* scheme was used, in which n-gram counts for the clean pages were subtracted from the corresponding n-gram counts for the raw text dumps. The differences between the two values approximate the n-gram counts that would be obtained from a boilerplate training corpus, consisting of the dirty text segments that have been deleted from the gold standard.

It has to be noted that differential n-gram counts are not fully consistent (because they cannot take the boundaries between dirty and clean text segments into account) and result in a *deficient* language model (in fact, estimated probabilities may be negative or greater than 1, and are clamped to the range $[0,1]$ after training). The StupidOS algorithm is founded on the assumption that this deficient language model is still a good approximation to the correct language model for dirty text segments, so that a comparison of its probabilities with those calculated by the clean model is meaningful.

3.3. Parameter tuning

The best n-gram size n and the optimal value for the interpolation parameter q were determined by minimisation of *cross-entropy* on the raw text dumps, calculated by 10-fold cross-validation.[5] Fig. 2 plots cross-entropy against the interpolation parameter

[5] Note that a straightforward minimisation of cross-entropy on the training data without cross-validation would lead to serious overfitting. The raw text dumps were used because they comprise

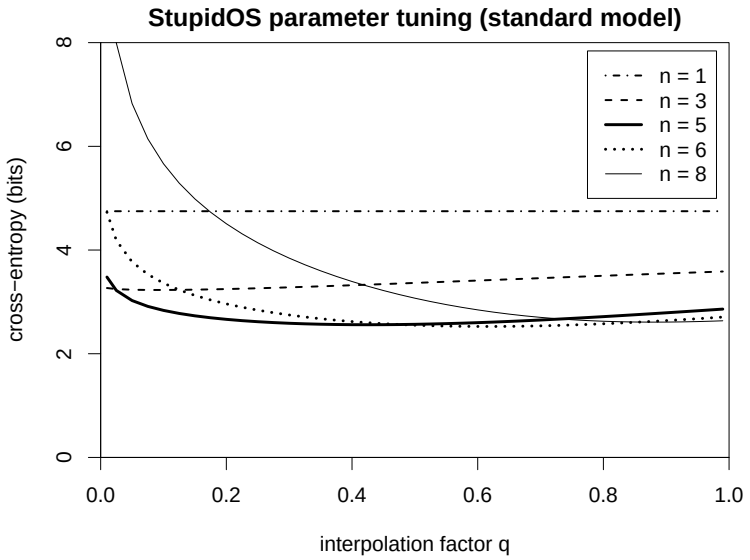

Figure 2. Parameter tuning for the n-gram language models.

q for various values of n. These graphs suggest $n = 6$ and $q = 0.6$ (a fairly strong interpolation where unigrams still have a relative weight of 0.03) as a good and robust choice, with a cross-entropy of 2.53 bits. The 6-gram model achieves considerably better cross-entropy than bigram or trigram models, while higher-order models (e.g. $n = 8$) require extremely strong interpolation, indicating a low degree of robustness.

However, evaluation of the full StupidOS system (cf. Sec. 4.) revealed that lower-order models achieve slightly better accuracy despite higher cross-entropy values (perhaps because of their better generalisation capabilities). After some unsystematic manual experimentation, a model with $n = 3$ and $q = 0.5$ was selected for the final implementation.

3.4. Extension to a multi-language model

A serious shortcoming of StupidOS (and supervised learning approaches in general) is the need for a sizeable training corpus of manually cleaned Web pages. Language models trained on English text will produce fairly random results when applied to another language (this is in part to be expected, since the dirty n-gram model is intended to learn keywords such as *copyright* or *click here* that are characteristic of boilerplate). A user wanting to apply the StupidOS algorithm to French or German Web pages, for instance, would have to construct his or her own gold standard to train the models.

Therefore, a simplified language model was designed in the hope of being able to overcome the language barrier and make the trained model usable at least for other European languages based on the latin alphabet. In the simplified model, a pre-processing step maps all letters to the character `a` and all digits to the character `0`. For example, the sentence *The answer is 42.* would be seen as *aaa aaaaaa aa 00.* by the simplified model. Since this mapping removes all lexical information (except for some weak statistics on word lengths), we hope that the simplified StupidOS algorithm can successfully be ap-

both the clean and the dirty training corpus.

	FIASCO		Devset	
	Precision	Recall	Precision	Recall
Baseline	84.16%	99.58%	86.70%	99.93%
SOS baseline	85.12%	94.11%	87.32%	96.47%
SOS standard	91.95%	91.25%	93.58%	90.92%
SOS simplified	90.17%	89.82%	91.72%	91.74%

Table 1. Overall evaluation results of the StupidOS system on two data sets, showing micro-averaged precision and recall, i.e. percentages of words across the entire data set.

plied e.g. to French and German Web pages after being trained on the English gold standard. [6]

While it is impossible to test our hypothesis directly for lack of suitable non-English reference data, the performance of the model on the English gold standard – which is expected to be lower than that of the standard model – can be seen as an indicator of the feasibility of this approach. Parameter tuning was performed by minimising cross-entropy as described in Sec. 3.3. Differences between trigram models and larger values of n were minimal, but the full evaluation showed higher-order models to be slightly better and more stable. Hence, $n = 6$ and $q = 0.4$ were chosen as model parameters, for a cross-entropy of 0.87 bits. The relatively small value of q probably reflects the lower degree of data sparseness faced by the simplified model.

4. Evaluation

4.1. Evaluation methodology

A full evaluation of the StupidOS system was carried out on the FIASCO gold standard, testing both the standard and the simplified algorithm as described in Sec. 3. A cross-validation technique was used, so that the language models applied to a given HTML page were trained on other texts from the gold standard only.

Because of the slow speed and high memory load of the official CLEANEVAL scoring software, a new evaluation script was implemented in Python, using the standard `difflib` package to compute an alignment between the automatically cleaned text and the gold standard. The Python script processes the 158 gold standard files within a few seconds and reports separate values for precision, recall and F-score [7] at the level of words. In addition, segmentation accuracy is measured by precision, recall and F-score for correctly placed segment breaks, irrespective of the segment types (`<p>`, `<h>` or `<l>`).

As a baseline, raw Lynx text dumps were used, which achieve surprisingly high F-score. A second baseline is given by the StupidOS framework *without* language models. While F-score turns out to be slightly lower than for raw text dumps, the StupidOS baseline

[6] An intermediate model that distinguishes between vowels and consonants was also tested. This model would map the example sentence above to *tta atttat at 00.* However, since this model may still be language dependent (because of different syllable patterns in the languages) and did not perform significantly better in the evaluation, it was abandoned in favour of the fully simplified model.

[7] F-score is the harmonic mean of precision and recall, and is often used as an overall evaluation measure that assigns good scores only if both precision *and* recall are high.

		word accuracy			*segmentation accuracy*		
		F-score	Precision	Recall	F-score	Precision	Recall
FIASCO	Baseline	85.32%	79.00%	**96.74%**	0.00%	0.00%	0.00%
	SOS baseline	84.06%	79.16%	94.11%	63.72%	54.84%	**90.41%**
	SOS standard	**87.63%**	**88.47%**	89.74%	**69.96%**	**71.22%**	75.14%
	SOS simplified	85.60%	85.35%	89.37%	66.60%	66.36%	75.42%
Devset	Baseline	83.74%	77.81%	96.19%	0.00%	0.00%	0.00%
	SOS baseline	85.18%	79.48%	**96.66%**	65.14%	57.01%	**89.69%**
	SOS standard	**88.60%**	**92.09%**	87.65%	**68.49%**	**77.42%**	69.27%
	SOS simplified	86.24%	85.00%	90.78%	66.75%	67.36%	75.32%

Table 2. Evaluation of StupidOS (SOS) on the FIASCO gold standard and the CLEANEVAL development set (Devset).

adds segment markers that are completely absent from the Lynx baseline.

Development and parameter tuning of the StupidOS algorithm have exclusively been based on the FIASCO gold standard. As a final test of the system, language models trained on the full gold standard were applied to the CLEANEVAL development set as completely unseen data, using the same evaluation metrics as above.

4.2. Results

A summary of the evaluation results is shown in Table 2. On the FIASCO data set, StupidOS achieves a significant improvement over the baseline, with an F-score of 87.63% vs. 85.32% (paired t-test across texts, $t = 1.976$, df $= 157$, $p = .0499$). In addition to this overall improvement, StupidOS trades off recall for precision, resulting in a considerably better precision of 88.47% vs. 79.00%. This difference is highly significant, and after correcting for random variation the net increase is at least 7.63% (paired t-test, 95% confidence interval). Surprisingly, results are even better for the completely unseen CLEANEVAL development set, with a chance-corrected net increase of at least 1.5% for F-score and 9.16% for precision.

Comparison with the StupidOS baseline shows the important role that the language models play in the system. Segmentation accuracy is decent, with F-scores close to 70%. There is no meaningful "uninformed" baseline for comparison, but it is obvious from Table 2 how the language models trade off recall for precision in this case as well.

All scores in Table 2 are *macro-averaged*, i.e. they represent averages over the scores for individual files, regardless of their sizes. *Micro-averaged* precision and recall scores, i.e. percentages of correct words in the entire data set, are even better (see Table 1). For instance, StupidOS achieves a micro-averaged precision of 91.95% vs. a baseline of 84.16% on the FIASCO data set, and 93.58% vs. 86.70% on the development set.

The simplified StupidOS algorithm is always significantly worse than the standard model, but it still achieves a substantial improvement over the baseline in precision. The small improvement in F-score is also significant on the development set, but not on the FIASCO gold standard. These results nurture hopes that the simplified model can at least be used as a rough-and-ready boilerplate removal tool for other European languages.

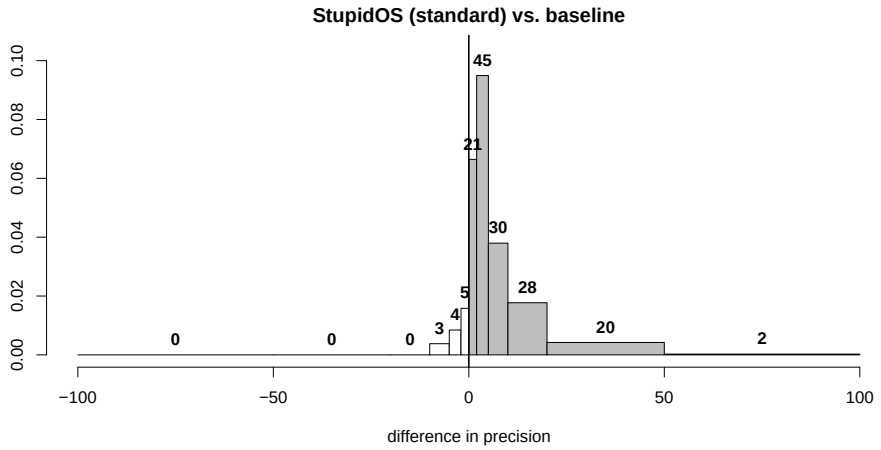

Figure 3. Precision differences

4.3. Processing speed and memory requirements

One advantage of the naive n-gram modelling approach taken by StupidOS is its low computational complexity, with parameter file sizes of only 1.5 MB for the standard model and 800 KB for the simplified model. Despite the inefficient Perl implementation, the system processes approximately 20 million words per hour on a standard server-class machine. [8] While running, it occupies less than 20 MB of working memory. This shows that the research prototype of StupidOS can readily be applied even to large Web corpora containing billions of words.

Clearly, the main performance bottleneck is the pure Perl implementation of the n-gram models. The StupidOS baseline system without language models processes up to 180 M words per hour, and a similar throughput can probably be achieved if the n-gram models are ported to compiled C code.

4.4. Discussion of the high-precision approach

It has already been argued that the main strength of StupidOS is the way in which it trades off recall for precision. While it achieves a moderate overall improvement in F-score, precision is considerably higher than for the baseline. In this way, a relatively high-quality Web corpus can be compiled, although some clean text from the original Web pages is lost. Given the huge size of the World Wide Web, at least for English and other major European languages, such a high-precision, low-recall approach will be appropriate for most applications of Web corpora.

Fig. 3 compares the precision achieved by StupidOS against baseline precision, for all files in the FIASCO data set. Except for a small number of files, the StupidOS algorithm always improves precision, and for about a third of the files, there is a substantial improvement of 10% or more. [9] Nonetheless, the StupidOS is not yet able to ensure

[8] 2 × dual-core AMD Opteron 285 @ 2.6 GHz, 16 GB RAM, normal processing load.

[9] A loss in precision is often due to character encoding problems, where e.g. Japanese Unicode characters are incorrectly converted to UTF-8 both by StupidOS and the baseline text dumps. Encoding is messed up in different ways, though, so that affected words cannot be aligned properly in the evaluation.

consistently high precision for all Web pages in a data set. Among the 158 files in the FIASCO gold standard, 17 are cleaned with a precision below 70%, and more than a third have precision below 90%.[10] Preliminary experiments suggest that larger files, and especially those with many large text segments, tend to have better precision than small files with many list items or very short paragraphs. Using linear models to predict boilerplate removal accuracy,[11] we selected a subset of 100 pages (with predicted precision $\geq 86\%$). Average precision on these files is 91.88% (compared to 88.47% for the entire data set), and there are only 5 files with precision below 70%.

5. Conclusion and outlook

In this paper, we presented the StupidOS algorithm, a naive approach to boilerplate removal based on character-level n-gram models and a manually cleaned training corpus. Evaluation on two data sets showed a significant improvement in boilerplate removal accuracy over the baseline. In addition, StupidOS trades off recall for precision, with less than 10% of boilerplate text remaining in its final output. Even the prototype implementation of StupidOS in pure Perl is lightweight and fast, so it can readily be applied to Web corpora in the gigaword range. A simplified version of the algorithm, which might be usable for other European languages without a need for additional training data, still achieves a significant improvement over the baseline precision.

There are still many ways in which the system can be improved. A first step would be to construct a reliably aligned version of the FIASCO gold standard, which may be possibe with the algorithm used for system evaluation in Sec. 4.1. Based on this new training corpus, more sophisticated language models could be developed, in particular making use of structural information from the original HTML code (which can be injected into the text dumps). Furthermore, a Viterbi search algorithm could be used to identify coherent blocks of clean or dirty text, rather than making an independent decision for each text segment (which may delete random segments from a connected discourse).

References

BARONI M. and BERNARDINI S. (Eds) (2006), *Wacky! Working papers on the Web as Corpus*, GEDIT, Bologna, Online version: `http://wackybook.sslmit.unibo.it/`.

BAUER D., DEGEN J., DENG X., HERGER P., GASTHAUS J., GIESBRECHT E., JANSEN L., KALINA C., KRÜGER T., MÄRTIN R., SCHMIDT M., SCHOLLER S., STEGER J., STEMLE E. and EVERT S. (2007), "FIASCO: Filtering the Internet by Automatic Subtree Classification, Osnabrück", in *this volume*.

KILGARRIFF A. and GREFENSTETTE G. (2003), "Introduction to the Special Issue on the Web as Corpus", in *Computational Linguistics*, n° 3, vol. 29.

MANNING C. D. and SCHÜTZE H. (1999), *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge, MA.

[10] For the simplified StupidOS model, there are 27 files below 70% and 69 files below 90%, while the raw Lynx dumps used as a baseline have 42 files below 70% and 94 files below 90%.

[11] Linear model for predicting precision from simple text statistics of the cleaned pages, with residual standard error of 13.76%. Significant factors were the number of text segments, average segment length and the proportion of text contained in very short segments.