FIASCO: Filtering the Internet by Automatic Subtree Classification, Osnabrück

Daniel Bauer, Judith Degen, Xiaoye Deng, Priska Herger, Jan Gasthaus, Eugenie Giesbrecht, Lina Jansen, Christin Kalina, Thorben Krüger, Robert Märtin, Martin Schmidt, Simon Scholler, Johannes Steger, Egon Stemle, Stefan Evert *
Institute of Cognitive Science, University of Osnabrück

Abstract

The FIASCO system implements a machine-learning approach for the automatic removal of boilerplate (navigation bars, link lists, page headers and footers, etc.) from Web pages in order to make them available as a clean and useful corpus for linguistic purposes. The system parses an HTML document into a DOM tree representation and identifies a set of disjoint subtrees that correspond to text blocks, headers or list items. Each block is then represented as a vector of linguistic, structural and visual features. A support vector machine classifier is used to distinguish between "clean" and "dirty" blocks. Dirty blocks are removed from the HTML tree before it is passed to the Lynx browser for conversion into plain text. The SVM classifier was trained and evaluated on a manually cleaned dataset of 158 English Web pages, the FIASCO gold standard.

Keywords : CLEANEVAL, Web as corpus, boilerplate, machine learning, support vector machines.

1. Introduction

The World Wide Web offers a unique possibility to gather large amounts of authentic and up-to-date language data from a wide variety of genres, and to build corpora containing billions of words of text with relatively little effort. [1] Such gigaword corpora are essential for studying lexical phenomena (such as collocations, or a lexicographic analysis of word meaning), and they have been shown to improve statistical NLP models drastically. If trained on huge amounts of data, even simple algorithms can produce high-quality results that outclass more sophisticated algorithms based on small training sets (Banko & Brill 2001).

There is one drawback, though: many of the Web pages collected by a crawler will be automatically generated pages, Web spam or duplicates of other pages. Accepting

* Institute of Cognitive Science, University of Osnabrück, {dbauer, jdegen, xiadeng, pherger, jgasthau, egiesbre, ljansen, ckalina, thkruege, rmaertin, martisch, sscholle, jsteger, estemle, severt}@uos.de

[1] From our experience working on this project, it appears to us that the effort involved in building a Web corpus has persistently been underestimated ...

such pages into a Web corpus would distort frequency counts in a way that no statistical model can account for. Fortunately, algorithms for filtering out such pages have been developed by the search engine and text mining communities, and can readily be applied. Even if the filters are effective, the remaining "good" pages will still contain a considerable amount of unwanted text. Navigation bars, link lists, banners, the headers and footers of page templates, disclaimers, copyright notices and, of course, the ubiquitous advertisements (collectively referred to as *boilerplate*) consist of canned or automatically generated text that should not be part of a linguistic corpus.

Although virtually every Web page contains some boilerplate text, sometimes amounting to more than half of the page content, there are no well-established algorithms for boilerplate removal and Web corpus builders have to rely on simple heuristic tools such as the PotaModule. [2] This paper describes our supervised learning approach to cleaning up Web pages, codenamed FIASCO, which has been developed for participation in the CLEANEVAL contest.

2. System architecture

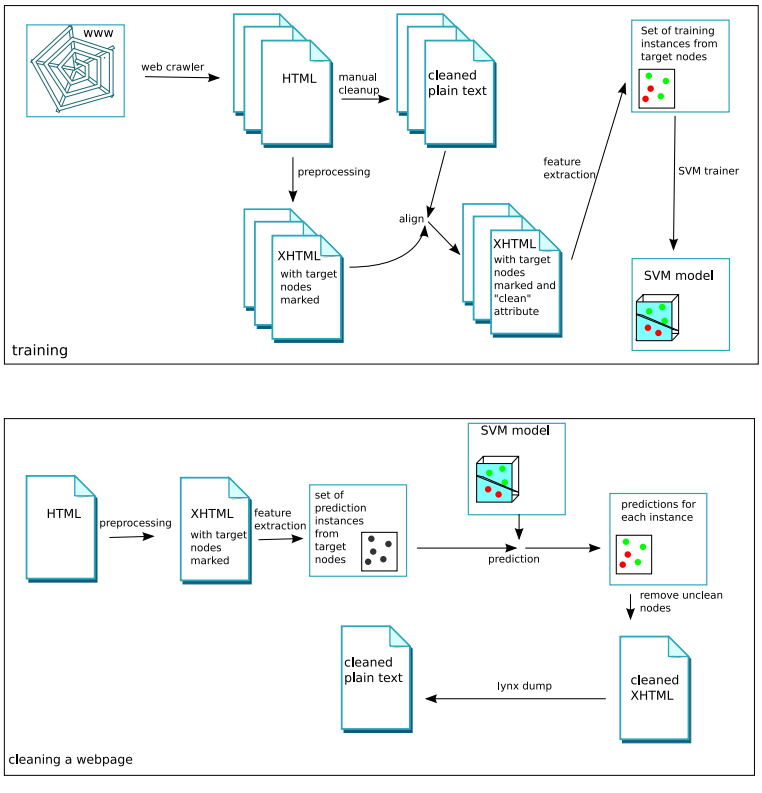

Figure 1. Training phase (top panel) and cleaning phase (bottom panel) of FIASCO.

Like all supervised learning approaches, FIASCO operates in two stages. In the first stage, a classifier is trained to discriminate between *dirty* (i.e., boilerplate) and *clean* subtrees in the DOM parse tree of an HTML file (Fig. 1, top panel). Sec. 2.3. describes this process in detail. In the second stage, the classifier is applied to the relevant subtrees

[2] `http://sslmitdev-online.sslmit.unibo.it/wac/post_processing.php`

of an unseen Web page (Fig. 1, bottom panel). Subtrees that are classified as dirty are removed from the parse tree. A description of this cleaning process is given in Sec. 2.4. In a final post-processing step, which is described in Sec. 2.5., the cleaned page is rendered as plain text. Notice that the HTML pages require some pre-processing in both stages, which is described in Sec. 2.1.

2.1. Preprocessing

HTML source files "in the wild" mostly do not correspond to the official W3C standard (Raggett *et al.* 1999) and, consequently, very robust parsers are needed to handle them. In both stages, i.e. training and classification, all HTML pages are therefore parsed and converted to valid XHTML using the open source parser TagSoup (Cowen 2004) – a SAX parser written in Java and designed to process even extremely malformed HTML sources. The XHTML files are also converted to UTF-8 encoding, if necessary. In addition, some rule based cleanup is done during preprocessing, viz. inline Javascript (marked with the `<script>` tag), style information and HTML comments are removed. Finally, in many web pages paragraphs are not contained in `<p>` elements but rather delimited by line breaks, marked with empty `<br/>` tags. These are replaced with proper paragraph elements in order to help our system find suitable text blocks for classification.

2.2. Identifying relevant subtrees

Our system is designed to distinguish between clean and dirty text on the basis of "text block"-like elements of a Web page. While it is relatively easy for humans to identify such elements in a rendered version of the page, i.e. the page as displayed in a Web browser, determining the precise parts of the original HTML document that correspond to such blocks is a non-trivial task. Using an in-memory DOM tree representation of the HTML document, the identification of text blocks amounts to finding subtrees with certain characteristics. The root node of such a subtree is called a *target node* and represents the automatically recognized text block in further processing. Unfortunately there is no one-to-one correspondence between certain types of HTML tags or elements and text blocks in the visual output. For example, the tags `<div>`, `<p>`, and `<td>` can all be used to create a visually separated block, but they can also be used for many other purposes.

After some manual experimentation, we decided on the following heuristic for identifying text blocks: a node in the DOM tree was accepted as a *target node* if 10% of its textual content (the concatentation of all text nodes in the subtree below the node) were contained in text nodes among its immediate children. This heuristic works reasonably well for detecting text elements at a paragraph-like granularity, giving an acceptable approximation to visually separated text blocks for our purposes.

2.3. Training

The supervised training of the FIASCO model consists of two stages: First, feature vectors (containing the features described in Sec. 3.) are created for each target node in the training corpus, yielding a set of labelled, real-valued vectors. The values of each feature (across all training instances) are then scaled linearly to the interval $[-1,1]$.

Second, the training vectors are passed to LIBSVM (Chang & Lin 2001) to build a support vector machine (SVM) model (Schölkopf & Smola 2002). The optimal model parameters were determined using cross-validation (as described in Sec. 4.1.).

2.4. Cleaning Web pages

Given a a raw HTML page to be cleaned, our system first pre-processes the page and identifies target nodes as described above. Afterwards, feature values for each target node are extracted and scaled (using the scaling factors determined during training) and then passed to the SVM model for prediction. The model returns the predicted class (either *clean* or *dirty*) of the node. All subtrees rooted in dirty nodes are pruned from the tree representation. The tree is then re-transformed into an HTML document and written to a file for further processing (cf. Sec. 2.5.).

2.5. Post-processing of cleaned Web pages

For the CLEANEVAL contest, output in the form of plain text files with text segments marked as `<p>` (paragraph), `<h>` (heading), or `<l>` (list item). Because no distinction is made between these different types of text blocks during the cleaning process, they had to be determined by the post-processing procedure. Automatically identified text blocks (cf. Sec. 2.2.) form the basic text segments. A simple heuristic is then used to determine segment type: If a node corresponds to (or is contained in) an HTML element that matches one of the paragraph types (`<p>` for plain paragraphs, `<li>` for list items, and `<h1>`, `<h2>`, `<h3>`, `<h4>` for headings) it is marked accordingly, otherwise it is marked as `<p>`. Further refinements are made after the document has been converted to plain text with the help of the Lynx text-mode browser.

3. Features

We implemented a small Python framework for extracting feature vectors from Web pages that have previously been converted to XHTML format. These features form the basic training set for the machine-learning classifier. For every target node indentified in the XHTML file, a number of features was extracted, then the feature values were scaled and converted to the data format required by LIBSVM. The features can be grouped into *linguistic* (Sec. 3.1.) and *structural* (Sec. 3.2.) features. Furthermore, we pursued a radically different approach based on *visual features* of the rendered Web pages (Sec. 3.3.).

3.1. Linguistic features

Linguistic features were extracted either from the text nodes immediately dominated by a target node, or from all text nodes in the complete subtree rooted in this node. Our framework implements the following features: (i) total length of text in the subtree; (ii) number of word tokens and word types; (iii) number of sentences and average sentence length; (iv) frequency of certain *keywords* indicative of clean and dirty text;[3] and (v)

[3] A *keyword* has to be significantly more frequent in one part of the training corpus (either clean or dirty) than in the other. In addition, keywords are required to be substantially more frequent than in general language, estimated by frequency counts on the *British National Corpus* (Aston & Burnard

statistics on the distribution of part-of-speech tags. [4]

The selection of features placed emphasis on being able to classify both nodes with short texts and nodes containing long text passages correctly. Thus, features such as the total length of the dominated text or the number of sentences immediately below the node allow the classifier to recognize long text blocks with "average-length" sentences, which would typically not contain boilerplate. On the other hand, the keyword-based features and the proportion of special symbols are intended to distinguish between short clean and short dirty text blocks.

3.2. Structural features

Structural features mainly provide information about tag or attribute densities in the subtree dominated by a target node. The intuition behind most of the structural features was that dirty text is likely to contain large numbers of images or external URLs, while clean nodes are often contained in paragraph, heading, or list elements. Some examples of structural features are: (i) depth of the target node within the DOM tree; (ii) whether target node is marked as a heading; (iii) whether target node is a `<p>` element; and (iv) the proportion of `<a>`, `<href>` and `<img>` tags below the target node.

3.3. Visual features

The function of textual components of a Web page cannot always be predicted by analyzing the HTML source code. In some cases, function follows form: The visual rendering of a Web page may reveal formatting subtleties that provide information about the purpose and function of its building blocks. Such information is not accessible to a purely structure-based or linguistic analysis. Since maps of local features – such as luminance, color contrast and texture contrast (Schumann 2006) – turned out not to provide much information about the function of text blocks, we decided to focus on the geometric properties of page regions corresponding to text blocks.

In order to recognize page regions corresponding to the target nodes identified by the FIASCO system (cf. Sec. 2.2.), we extended the Firefox rendering engine with an add-on that changes attributes of specially marked text elements in the DOM tree in such a way that they are displayed as bright purple blocks. Visual features were then extracted from the corresponding regions of a normally rendered version of the page, and fed back into FIASCO as a table mapping target nodes to feature vectors.

1998). Sets of keywords were automatically extracted from the text version of our gold standard after tokenization with the TreeTagger tokenizer (Schmid 1994). They were ranked by *keyness*, a conservative estimate for the log odds ratio between their relative frequencies in clean and dirty text, and the highest-ranking keywords were used in the FIASCO system.

[4] The textual content of the subtree below each target node was extracted and tagged with Tree Tagger (Schmid 1994), using the standard English parameter file provided with the system. Afterwards, the proportions of *closed class* words, *open class* words, and *unknown words* (word forms not listed in the tagger lexicon) were computed for each target node. The intuition behind these features is that a higher proportion of unknown words as well as the absence of *closed class* words are strong indicators that the text belongs to a dirty node. Similarly, a high concentration of "special symbols" (words tagged `SYM`, various types of punctuation, as well as foreign words tagged `FW`) within a node is often characteristic of boilerplate, i.e. dirty text.

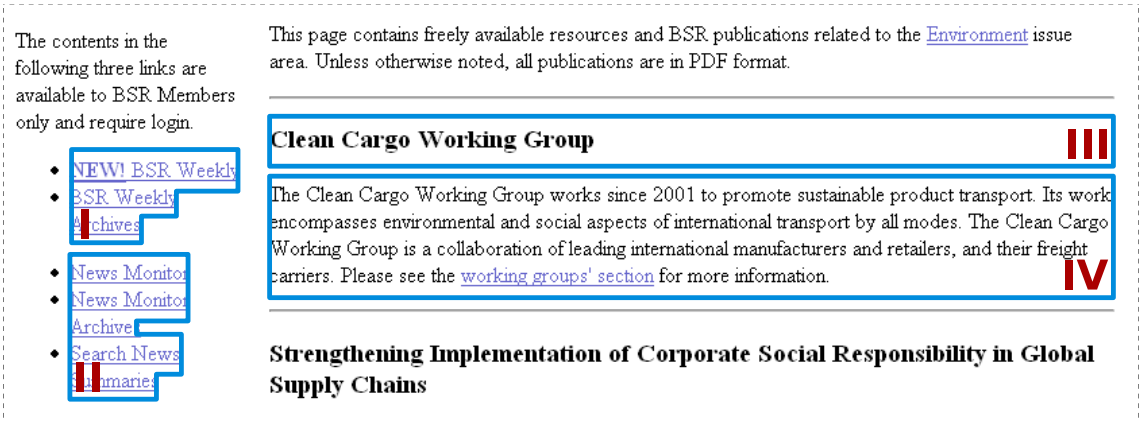

Figure 2. Sample from a Web page. Page regions considered by the algorithm for visual feature extraction are marked with blue lines.

Region	Solidity I	Solidity II	Solidity III	Relative Width	Headline
I	0.71	1.35	0.12	0.53	0.00
II	0.88	1.30	0.09	0.60	0.00
III	0.97	0.00	0.61	1.00	0.51
IV	0.99	0.00	0.61	1.00	0.00

Table 1. Feature values for regions highlighted in Fig. 2.

We made the following assumptions about characteristic properties of regions containing clean text: [A] Informative running text in the main body of a Web page occupies a major portion of the screen, i.e. the text regions are relatively *wide*. In contrast, text regions belonging to uninformative boilerplate and advertisements are usually confined to *narrow* columns or rows at the edges of the page. [B] Running text shows a relatively *uniform* distribution of line widths, in constrast to list items. Only the last line of each paragraph may deviate considerably from the average width of the entire text block. [C] Headings are clearly separated text regions that are only large enough to hold a single line of text (possibly in a very large font).

In order to determine geometric properties of text regions, the page image is convolved with a mean filter or a Gaussian kernel of appropriate size. This causes lines that are close to one another to coalesce into a single connected blob, while text blocks separated by blank lines remain separated. From these blobs, we calculated the five measures below to quantify our intuitions on the properties of informative running text. Most of the processing was done with Intel's Open Computer Vision library (Bradski 2000).

Solidity I: The polygon enclosing a region of running text is compact, i.e. very similar to a rectangle. Therefore, the ratio between the area of the enclosing polygon and the area of the minimal bounding rectangle (MBR) should be close to one for running text. **Solidity II:** A similar measure is defined by the number of vertices of the enclosing polygon, divided by the region's height in line units. **Solidity III:** More specific than Solidity I, this measure only computes the average distance of polygon vertices to the center of the MBR along the x-axis. **Relative Width:** Since running text is exptected to occur in relatively wide columns, this measure compares the width of the current region

of interest to the widest region on the Web page. **Headline:** A headline should not be much higher than the average headline height. Therefore, the relative difference between a region's height and the average headline height was used as an additional feature.

Table 1 lists the feature values calculated for the four regions of interest highlighted in the Web page excerpt shown in Fig. 2. Amongst the solidity measures, II and III most clearly discriminate between running body text (regions III, IV) and boilerplate (regions I and II). Relative width is useful for distinguishing blocks of running text from list items. Combining these two measures with the headline feature allows for a clear separation of running body text, headlines, and itemized or enumerated lists.

4. Supervised training of the classifier

4.1. SVM parameter optimisation

We used the parameter optimization tool included in the standard LIBSVM distribution, which performs a *grid search* for optimal parameter values C and γ of the radial basis function (RBF) kernel. The accuracy on a subset of our gold standard was determined by 3-fold cross-validation for each set of parameters. Starting from 84.5%, the highest accuracy of 89.1% was achieved for parameter values $C = 32$ and $\gamma = 8$, after testing around 110 different combinations of parameter values.

Systematic parameter tuning was also performed for other kernels (linear, polynomial and sigmoid), using the same optimization procedure. The best result was achieved by the polynomial kernel (87.5% accuracy), but remains below the accuracy of the RBF kernel. For comparison, a decision tree classifier [5] was applied to the same data set. However, classification accuracy was rather poor. Even with additional bagging and boosting, accuracy only reached 80%, indicating that SVM are indeed the most suitable type of classifier for this task.

4.2. Evaluation of the classifier

In order to evaluate the overall classification accuracy that can be achieved with the features we implemented, as well as the performance of different subsets of features, we trained SVM classifiers using an RBF Kernel (with the optimal parameter settings $\gamma = 8.0$ and $C = 32.0$ determined in Sec. 4.1.). The results of a 10-fold cross-validation are shown in Table 2. [6] In addition to overall accuracy, precision and recall for the identification of clean target nodes are shown in the table. For the goal of building a high-quality Web corpus (rather than obtaining complete data from every Web page), these may be the most relevant evaluation metrics. While the part-of-speech features perform quite well on their own, they have an adverse effect on the overall result when used in conjunction with all other features. Visual features on the other hand improve the overall result, while performing worse than part-of-speech features when tested separately.

[5] `http://www.cs.utah.edu/~hal/FastDT/index.html`

[6] Unfortunately, the Solidity III feature was broken in the version of our program submitted to the CLEANEVAL and evaluated here. When this bug was fixed, the visual features achieved a 84% classification accuracy (precision 0.68, recall 0.85, F-measure 0.75).

Features	Accuracy	Precision	Recall	F-Measure
All	91.13%	0.90	0.80	0.85
All\{POS}	**92.50%**	**0.90**	**0.86**	**0.88**
All\{visual}	90.01%	0.87	0.80	0.83
All\{POS, visual}	90.17%	0.86	0.82	0.84
Visual only	79.50%	0.76	0.51	0.61
POS only	83.67%	0.80	0.63	0.71
Only POS and Visual	86.69%	0.82	0.74	0.78

Table 2. Model performance using different feature subsets.

5. Gathering training data

For training the FIASCO classifier, a substantial amount of hand-annotated data are needed. The development data set provided by the CLEANEVAL organizers contained only 59 English and 51 Chinese pages, and the manually cleaned versions of these pages showed some inconsistencies and problems with the annotation guidelines. We therefore downloaded over 300 Web pages using Google queries with random combinations of seed terms from different domains. We manually preselected 158 pages, which contained at least one paragraph of human-readable running text. For each page, we created a text dump using the text-based Web browser Lynx. The pages were then manually cleaned by two human annotators, who also marked paragraphs, list items and headings using single `<p>`, `<l>` or `<h>` tags, respectively. Cleanup and annotation were based on a refined version of the official CLEANEVAL guidelines, and annotators were provided with a graphical tool in order to facilitate the task. Minimal edit distance between the two annotations for each page was computed, showing an average agreement of 91.75%. The two versions of each text were manually merged by a third annotator for the final version of our gold standard.

5.1. Alignment to HTML target nodes

Unfortunately, we made the mistake of using the same annotation strategy as the official CLEANEVAL data set. This meant that all internal structural information of the HTML document was lost in the manual cleanup process (since the cleanup started from text dumps of the pages) and the gold standard was therefore not directly usable as training data for our machine-learning algorithm. In order to identify sets of *clean* and *dirty* target nodes, the paragraphs of the manually cleaned plain text files had to be aligned to subtrees of nodes in the HTML document, by calculating Dice coefficients for the word overlap between each paragraph and the textual content of each HTML subtree. If this process found a HTML subtree that was sufficiently similar to a clean paragraph, its root node was marked as clean. The information was then propagated up and down the tree to previously selected target nodes (see Sec. 2.2.): Target nodes contained in a clean subtree as well as those dominating more clean than non-clean nodes were marked as clean. All remaining target nodes were marked as dirty, and the two sets of target nodes were then passed on to the feature extraction process.

The accuracy that can be achieved by our machine-learning approach crucially depends

Features	Accuracy	Precision	Recall	F-Measure
All	95.66	0.97	0.92	0.95
All\{POS}	**95.76**	**0.95**	**0.94**	**0.95**
All\{visual}	94.13	0.94	0.91	0.93
All\{POS, visual}	93.13	0.93	0.90	0.91
Visual only	79.91	0.83	0.64	0.72
POS only	82.42	0.85	0.69	0.76
Only POS and Visual	88.81	0.90	0.82	0.86

Table 3. Performance of classifier trained on gold standard files for which the automatic alignment algorithm worked well.

on the quality of the target node identification and their alignment with the gold standard (cf. Fig. 1). Therefore, we undertook a small-scale manual evaluation of these stages. In our gold standard, we found 8 files where only 1 to 4 target nodes had been marked in the DOM tree, compared to 17 up to 127 segments in the corresponding manually cleand text file. It is obvious that no sensible alignment can be achieved for these files. Of the remaining 150 gold standard files, a third aligned all text segments in the manually cleaned version to nodes in the XHTML file (leading to correctly identified clean target nodes). Another third achieved relatively good alignment rates over 80%, while the final third had poor alignment quality and may not be usable at all.

In order to test the influence of alignment quality, we repeated the evaluation of Sec. 4.2., using only the 117 best-aligned files. All other parameters were exactly the same, i.e. we used a RBF Kernel with $\gamma = 8.0$ and $C = 32.0$. The results of a 10-fold cross-validation in Table 3 show a marked improvement in overall accuracy, confirming our hypothesis that good alignment is crucial for the FIASCO system. Interestingly, the POS features no longer seem to have the detrimental effect that was observed in the first evaluation.

6. Evaluation of the FIASCO

We evaluated the overall performance of our system by comparison with the manually cleaned gold standard (see Sec. 5.). Precision, recall and F-score were calculated at word level, using a fast Python implementation (Evert 2007) based on the `difflib` package. The evaluation was carried out by 5-fold cross-validation, i.e. we split our 158 gold standard files randomly into five sets, using one of them as a test set in each fold to evaluate a classifier trained on the remaining four sets. We used all features described in Sec. 3. and the optimal parameter settings for the SVM classifier determined in 4.1. The mean precision over all folds was 87.63%, with a recall of only 51.87%, yielding an F-score of 65.16%. While this F-Score falls far below the baseline of 85.32% (given by plain Lynx text dumps, without any cleanup), we have been able to improve precision considerably from its baseline of 79.00%.

While we are pleased with the high precision achieved by FIASCO (which means that only a small amount of "dirty" text is left in the automatically cleaned pages), the surprisingly low recall merits further investigation. We found that it was caused by a small number of pages with recall below 1%. Manual inspection of these pages revealed that

most of them were either empty after the pre-processing step (see Sec. 2.1.), or only a single target node had been identified by our system, typically encompassing the entire <body> element. Unsurprisingly, this node was then classified as dirty. If such pages are discarded, average recall increases to 68.94% with a precision of 87.05%, corresponding to an F-score of 76.44%. Although the F-score is still below the baseline, these results show a reasonable trade-off between precision and recall that seems warranted in the context of Web corpora.

7. Conclusion

Our experiments with the FIASCO system have shown that a supervised machine learning approach to the problem of boilerplate removal is feasible, and that high classification accuracy for "clean" and "dirty" text blocks can be achieved. The weakest points of our approach are the automatic identification of "target nodes" in the HTML tree corresponding to text blocks or paragraphs, and the alignment between these target nodes and manually cleaned text in the gold standard.

While the overall evaluation results show an F-score well below the baseline given by plain text dumps of the HTML pages, high precision can be achieved ($> 87\%$ compared to a baseline of 79%). In the context of Web corpora, which are often aimed at an opportunistic collection of large amounts of text, a boilerplate removal strategy with high-precision seems to be justified, even if recall is fairly low.

One problem of supervised techniques is that they are not language-independent in the sense that an expensive, manually annotated training corpus is needed for every language to which they are applied. A version of our classifier using only visual and structural features of Web pages may have the potential to overcome these restrictions and has been applied to the Chinese evaluation set in the CLEANEVAL contest.

References

ASTON G. and BURNARD L. (1998), *The BNC Handbook*, Edinburgh University Press, Edinburgh, See also the BNC homepage at http://www.natcorp.ox.ac.uk/.

BANKO M. and BRILL E. (2001), "Scaling to very very large corpora for natural language disambiguation", in *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*.

BRADSKI G. (2000), "The OpenCV Library", in *Dr. Dobb's Journal of Software Tools*, n° 11, vol. 25.

CHANG C.-C. and LIN C.-J. (2001), *LIBSVM: a library for support vector machines*, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

COWEN J. (2004), *TagSoup*, http://mercury.ccil.org/ cowan/XML/tagsoup/.

EVERT S. (2007), "StupidOS: A high-precision approach to boilerplate removal", in *this volume*.

RAGGETT D., LE HORS A. and JACOBS I. (1999), "HTML 4.01 Specification", in *W3C Recommendation REC-html401-19991224, World Wide Web Consortium (W3C), Dec*.

SCHMID H. (1994), "Probabilistic Part-of-Speech Tagging Using Decision Trees", in *Proceedings of the International Conference on New Methods in Language Processing (NeMLaP)* : 44–49.

SCHÖLKOPF B. and SMOLA A. J. (2002), *Learning with Kernels*, MIT Press.

SCHUMANN F. (2006), *Integration of different features in guiding eye-movements*, Master's thesis, University of Osnabrück.