Victor: the Web-Page Cleaning Tool

Miroslav Spousta, Michal Marek, Pavel Pecina

Institute of Formal and Applied Linguistics,
Charles University, Prague, Czech Republic
{spousta,marek,pecina}@ufal.mff.cuni.cz

Abstract

In this paper we present a complete solution for automatic cleaning of arbitrary HTML pages with a goal of using web data as a corpus in the area of natural language processing and computational linguistics. We employ a sequence-labeling approach based on Conditional Random Fields (CRF). Every block of text in analyzed web page is assigned a set of features extracted from the textual content and HTML structure of the page. The blocks are automatically labeled either as *content segments* containing main web page content, which should be preserved, or as *noisy segments* not suitable for further linguistic processing, which should be eliminated. Our solution is based on the tool introduced at the CLEANEVAL 2007 shared task workshop. In this paper, we present new CRF features, a handy annotation tool, and new evaluation metrics. Evaluation itself is performed on a random sample of web pages automatically downloaded from the Czech web domain.

1. Introduction

The idea of using "web as a corpus" has been very attractive for many researchers in computational linguistics, natural language processing, and related areas, who would really appreciate having access to such amount of data. The traditional way of building text corpora is a very expensive and time-consuming process and does not satisfy current requirements of modern methods. By automatic downloading of textual data directly from the web we can build extremely large corpus with relatively low cost and within short period of time.

Creating such a corpus comprises two steps: a) *web crawling* – automatic browsing the web and keeping a copy of visited pages and b) *cleaning* the pages to be included in the corpus. While there is a number of suitable web crawlers available (e.g. Heritrix[1], Holmes[2] or Egothor (Galamboš, 2006)), challenging task to clean up acquired web pages remains. Apart from main (linguistically valuable) content, a typical web page contains also material of no linguistic interest, such as navigation bars, panels and frames, page headers and footers, copyright and privacy notices, advertisements and other uninteresting data (often called *boilerplate*). The general goal is to detect and remove such parts from an arbitrary web page.

In this paper we describe a complete set of tools that enables transformation of a large number of web pages downloaded from the Internet into a corpus usable for NLP and computational linguistic research. The basis of our solution is the web-page cleaning tool first introduced at the CLEANEVAL 2007 shared task workshop (Marek et al., 2007). In order to approach structure of traditional corpora, we significantly modified the cleaning requirements and restricted the set of possible labels to *text* and *header* for *content segments* to be preserved and *other* for *noisy segments* to be eliminated.

First, we review the cleaning algorithm and its features, then we introduce an annotation tool developed for our purpose to prepare data for training and evaluation, and finally we present several experiments and their results. Our focus on the Czech language (mainly in the evaluation section) is induced by an intention to create a large Czech corpus, comparable to the largest corpora currently available. Needless to say, our tools are language independent and can be used for any language.

2. Related Work

Most of the work related to web page cleaning originated in the area of web mining and search engines, e.g. (Cooley et al., 1999) or (Lee et al., 2000). In (Bar-Yossef and Rajagopalan, 2002), a notion of pagelet determined by the number of hyperlinks in the HTML element is employed to segment a web page; pagelets whose frequency of hyperlinks exceeds a threshold are removed. (Lin and Ho, 2002) extract keywords from each block content to compute its entropy, and blocks with small entropy are identified and removed. In (Yi et al., 2003) and (Yi and Liu, 2003), a tree structure is introduced to capture the common presentation style of web pages and entropy of its elements is computed to determine which element should be removed. In (Chen et al., 2006), a two-stage web page cleaning method is proposed. First, web pages are segmented into blocks and blocks are clustered according to their style features. Second, the blocks with similar layout style and content are identified and deleted.

Many new approaches to web page cleaning were encouraged by the CLEANEVAL 2007 contest[3] organized by ACL Web as Corpus interest group. Competitors used heuristic rules as well as different machine learning methods, including Support Vector Machines (Bauer et al., 2007), decision trees, genetic algorithms and language models (Hofmann and Weerkamp, 2007). Although methods are fundamentally different, many of them employ similar set of mostly language-independent features such as average length of a sentence or ratio of capitalized words in a page segment.

[1] http://crawler.archive.org/

[2] http://www.ucw.cz/holmes/

[3] http://cleaneval.sigwac.org.uk/

3. Victor the Cleaner

3.1. System Overview

Our system for web page cleaning, first described in (Marek et al., 2007), is based on a sequence labeling algorithm with CRF++[4] implementation of Conditional Random Fields (Lafferty et al., 2001). It is aimed at cleaning arbitrary HTML pages by removing all text except headers and main page content. Continuous text sections (sections not including any HTML tags) are considered a single *block* that should be marked by a label as a whole.
The cleaning process consists of several steps:

1) Filtering invalid documents

Text from input documents is extracted and simple n-gram based classification is applied to filter out documents not in a target language (Czech in our case) as well as documents containing invalid characters (caused mainly by incorrect encoding specified in HTTP or HTML header).

2) Standardizing HTML code

The raw HTML input is passed through Tidy[5] in order to get a valid and parsable HTML tree. During development, we found only one significant problem with Tidy, namely interpreting JavaScript inside the `<script>` element, and employed a simple workaround for it in our system. Except for this particular problem which occurred only once in our training data, Tidy has proved to be a good choice.

3) Precleaning

Afterwards, the HTML code is parsed and parts that are guaranteed not to carry any useful text (e.g. scripts, style definitions, embedded objects, etc.) are removed from the HTML structure. The result is valid HTML code.

4) Text block identification

In this step, the precleaned HTML text is parsed again with a HTML parser and interpreted as a sequence of text *blocks* separated by one or more HTML tags. For example, the snippet `"<p>Hello <b>world</b>!</p>"` would be split into three blocks, `"Hello"`, `"world"`, and `"!"`. Each of the blocks is then a subject of the labeling task and cleaning.

5) Feature extraction

In this step, a feature vector is generated for each block. The list of features and their detailed description is presented in the next section. All features must have a finite set of values[6]. The mapping of integers and real numbers into finite sets was chosen empirically and is specified in the configuration. Most features are generated separately by independent modules. This allows for adding other features and switching between them for different tasks.

6) Learning

Each *block* occurring in our training data was manually assigned one of the following labels: *header*, *text* (*content blocks*) or *other* (*noisy blocks*).

[4]http://crfpp.sourceforge.net/

[5]http://tidy.sourceforge.net/

[6]This is a limitation of the CRF tool used.

The sequence of feature vectors including labels extracted for all blocks from the training data are then transformed into the actual features used for training the CRF model according to offset specification described in a template file.

7) Cleaning

Having estimated parameters of the CRF model, an arbitrary HTML file can be passed through steps 1–4, and its blocks can be labeled with the same set of labels as described above. These automatically assigned labels are then used to produce a cleaned output. Blocks labeled as *header* or *text* remain in the document, blocks labeled as *other* are deleted.

3.2. Feature Descriptions

Features recognized by the system can be divided by their scope into three subsets: features based on the HTML markup, features based on textual content of the blocks, and features related to the document.

Markup-based Features

container.p, container.a, container.u, container.img, container.class-header, container.class-bold, container.class-italic, container.class-list, container.class-form

For each parent element of a block, a corresponding *container.** feature will be set to 1, e.g. a hyperlink inside a paragraph will have the features *container.p* and *container.a* set to 1. This feature is especially useful for classifying blocks: For instance a block contained in one of the `<hx>` elements is likely to be a header, etc. The *container.class-** features refer to classes of similar elements rather than to elements themselves.

split.p, split.br, split.hr, split.class-inline, split.class-block

For each opening or closing tag encountered since the last block, we generate a corresponding *split.** feature. This is needed to decide, whether a given block connects to the text of the previous block (classified as *continuation*) or not. Also, the number of encountered tags of the same kind is recorded in the feature. This is mainly because of the `<br>` tag; a single line break does not usually split a paragraph, while two or more `<br>` tags usually do. The *split.class-** features again refer to classes of similar elements.

Content-based Features

char.alpha-rel, char.num-rel, char.punct-rel, char.white-rel, char.other-rel

These features represent the absolute and relative counts of characters of different classes (letters, digits, punctuation, whitespace and other) in the block.

token.alpha-rel, token.num-rel, token.mix-rel, token.other-rel token.alpha-abs, token.num-abs, token.mix-abs, token.other-abs

These features reflect counts distribution of individual classes of tokens[7]. The classes are words, numbers, mixture of letters and digits, and other.

[7]Tokens being sequences of characters separated by whitespace for this purpose.

sentence.count

Number of sentences in a block. We use a naive algorithm basically counting periods, exclamation marks and question marks, without trying to detect abbreviations. Given that the actual count is mapped into a small set of values anyway, this does not seem to be a problem.

sentence.avg-length

Average length of a sentence, in words.

sentence-begin, sentence-end

These identify text blocks that start or end a sentence. This helps recognizing headers (as these usually do not end with a period) as well as continuation blocks (*sentence-end=0* in the previous blocks and *sentence-start=0* in the current block suggest a continuation).

first-duplicate, duplicate-count

The *duplicate-count* feature counts the number of blocks with the same content (ignoring white space and non-letters). The first block of a group of twins is then marked with *first-duplicate*. This feature serves two purposes: On pages where valid text interleaves with noise (blogs, news frontpages, etc), the noise often consists of some phrases like "read more...", "comments", "permalink", etc, that repeat multiple times on the page.

regexp.url, regexp.date, regexp.time

While we try to develop a tool that works independently of the human language of the text, some language-specific features are needed nevertheless. The configuration defines each *regexp.** feature as an array of regular expressions. The value of the feature is the number of the first matching expression (or zero for no match). We use two sets of regular expressions: to identify times and dates and URLs.

div-group.word-ratio, td-group.word-ratio

The layout of many web pages follows a similar pattern: the main content is enclosed in one big `<div>` or `<td>` element, as are the menu bars, advertisements etc. To recognize this feature and express it as a number, the parser groups blocks that are direct descendants of the same `<div>` element (`<td>` element respectively). A direct descendant in this context means that there is no other `<div>` element (`<td>` element respectively) in the tree hierarchy between the parent and the descendant. For example in this markup

```
<div> a <div> b c </div> d <div> e
f </div> g </div>
```

the *div-group*s would be (a, d, g), (b,c) and (e, f). The *div-group.word-ratio* and *td-group.word-ratio* express the relative size of the group in number of words. To better distinguish between groups with noise (e.g. menus) and groups with text, only words not enclosed in `<a>` tags are considered.

langid1, langid2

These new features represent a probability that a text block is in given language (Czech in our experiment). We used our own implementation of two Language ID approaches: (Beesley, 1988) and (Cavnar and Trenkle, 1994).

Document-related features

position

This feature reflects a relative position of the block in the document (counted in blocks, not bytes). The rationale behind this feature is that parts close to the beginning and the end of documents usually contain noise.

document.word-count, document.sentence-count, document.block-count

This feature represents the number of words, sentences and text blocks in the document.

document.max-div-group, document.max-td-group

The maximum over all *div-group.word-ratio* and a maximum over all *td-group.word-ratio* features. This allows us to express "fragmentation" of the document – documents with a low value of one of these features are composed of small chunks of text (e.g. web bulletin boards).

4. The Annotation Tool

In order to enable fast and efficient annotation of the web page text blocks we developed a new annotation tool. Our aim was to offer a possibility to see the web page in a similar fashion to regular web browsing. This greatly simplifies the process of selection of the most important parts of given web page and distinguishing important text passages from other page sections.

Our annotation tool is a client–server based application using common web browser and JavaScript for the web page annotation on the client side and PHP based server application for serving pages to the client and storing current user annotation judgments.

The tool accepts either a list of HTML pages for annotation or a list of URLs to be downloaded and annotated. A simple pre-processing is applied for every web page before it can be annotated: all JavaScript is stripped and links are disabled so that annotator cannot accidently exit current web page.

The annotation process is quite straightforward (see Figure 1): user chooses a label selecting appropriate button and marks text blocks by clicking on the beginning and end of the text section to be marked. Different colors are used for every annotation label. Current annotation mark-up is stored on the server and can be easily retrieved and merged into the original HTML document when the annotation is finished.

We found that using a web browser for annotation significantly improves the annotation speed compared to using word processor or simple text based selection tool. The major speed up is due to the fact that not all the blocks must be

Figure 1: The annotation tool: browser window is split into two parts: narrow upper frame is used for annotation control, lower frame contains the page to be annotated. Current annotation is shown using different colors for every label.

judged and annotated — remaining unannotated blocks are implicitly classified as *other*. Our volunteer annotator was able to achieve speed of 200 web pages per hour.

5. Evaluation

5.1. Data preparation

In order to perform the cleaning task, we have to train the Conditional Random Fields algorithm on the data from pre-annotated web pages. For training and evaluation purposes, we selected a random sample of 2 000 web pages from the data set downloaded using the Egothor search engine robot (Galamboš, 2006) from the Czech web domain (.cz).

Large proportion of downloaded pages contains only HTML mark-up with none or very small amount of textual information (root frames, image gallery pages, etc.). In addition, many pages use invalid character encoding or contain large passages in a language different from our target language. In order to exclude such pages from further processing, we apply a Language ID filter. Each page is assigned a value which can be interpreted as a probability of the page being in Czech. Pages not likely to be in Czech are discarded. We used our own implementation of Language ID methods by Beesley (1988) and Cavnar and Trenkle (1994). Out of 2 000 web pages, only 907 were accepted as reasonable documents containing non-trivial amount of Czech text.

All documents were annotated using our HTML annotation tool described in the previous section. We provided only short annotation guidelines, discouraging a markup of short, incomplete sections of the text (product descriptions, lists of items, discussions) and only marking headlines belonging to already selected text sections. All non-annotated text blocks are considered to be labeled as *other*.

According to the annotation, only 271 (29.9%) documents contained text blocks with useful content (text and headers) to be preserved. Complete overview of the label distribution can be found in Table 1.

label	*count*	%
header	1 009	1.14
text	5 571	6.32
other	81 528	92.53
total	88 108	100.00

Table 1: Label distribution in the development data set.

5.2. Experiments and Results

Following our experience from the CLEANEVAL 2007 we found that computation of Levensthein algorithm for evaluation of cleaning results is usually very expensive. Our approach of labeling consequent text blocks suggests an *accuracy* as a measure of success for our task – the ratio of correctly assigned labels. If we do not want differentiate between blocks labeled as *text* or *header* (they are equally good for our purposes and we would like them to be included in our corpus) we can use also *unlabeled accuracy*.

In our first experiment we used 271 manually annotated pages containing at least one content block (labeled either as *text* or *header*). Running a 10-fold cross-evaluation on such data we were able to achieve accuracy of 91.13% and unlabeled accuracy of 92.23%. This number, however, does not tell much about quality of cleaning because of the discrepancy in proportion of content (*text*, *header*) and noise

(*other*) blocks in our data (see table 1) which could be expected also in real pages downloaded from the web. A trivial algorithm assigning *other* label to all blocks performs with accuracy even higher (92.53)%.

Precision and Recall

In such cases, using *precision* and *recall* measures would be more appropriate. We do not differentiate between blocks labeled as *text* or *header* (they are equally good for our purposes and we would like them to be included in our corpus) and define *precision* and *recall* as follows:

$$Precision = \frac{TH_{correct}}{TH_{labeled}}$$

$$Recall = \frac{TH_{correct}}{TH_{annotated}}$$

where $TH_{correct}$ refers to a number of correctly labeled content blocks [8], $TH_{labeled}$ is the total number of labeled content blocks and $TH_{annotated}$ is the total number of all blocks annotated as content blocks (*text* or *header*).
Precision and recall scores of our first experiment are shown in the first column of the table bellow. We can expect the pages to be cleaned with 80.75% precision and 79.88% recall, i.e. 19.25% of blocks in the cleaned data are noise and we miss 21.12% of content blocks that should be preserved.

using LangID	no	yes
Text/Header/Other		
Accuracy	91.13	90.82
Text+Header/Other		
Accuracy	92.23	91.84
Precision	80.75	83.80
Recall	79.88	72.95

Table 2: Effect of Language ID features.

Language ID features

In the next experiment we evaluated the Language ID features newly used by the CRF component of our system and representing probability that a text block is in given language (see section 3.2.). As it can be seen in the second column of Table 2, using these features we were able to increase the precision up to 83.8%.

Balancing Precision and Recall

The huge number of texts available on the web even for relatively rare languages such as Czech, enables us to focus on acquisition of high quality data only. In other words, we prefer high-precision cleaning procedure to the high-recall one.
While CRF algorithm does not offer a direct method to fine-tune precision and recall trade-off, we propose an alternative approach to achieve this. For every block, it is possible to obtain marginal probability assigned to all possible labels. In common sequence-labeling scenario, the label with highest probability wins, not matter what is a distribution of other labels' probabilities. In order to achieve higher precision, we only allow *text* and *header* labels to win if probability of *other* label is under given threshold. Figure 2 illustrates, that we are really able to achieve arbitrary precision by giving preference to the *other* label.

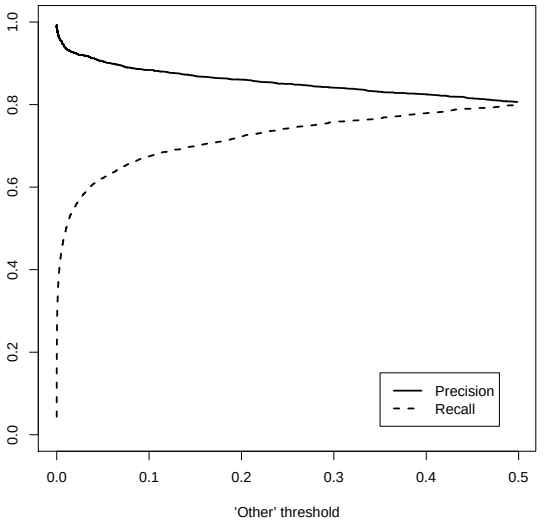

Figure 2: Precision and recall graph obtained by setting the threshold value of *other* label in the interval [0, 0.5]. Default value of the threshold is 0.5.

training data size (documents)	271	907
Text/Header/Other		
Accuracy	91.13	95.92
Text+Header/Other		
Accuracy	92.23	96.15
Precision	80.75	74.78
Recall	79.88	66.95

Table 3: Results of 10-fold cross-evaluation on 271 annotated documents (containing at least one block marked as *text* or *header*) and entire set of 907 annotated documents.

The last experiment we performed was a comparison of two systems: the one as in the first experiment trained on the 271 manually annotated pages containing at least one content block (labeled either as *text* or *header*) and the other (more real) one trained on all 907 manually annotated pages that passed the Language ID test. The performance of these two systems can be compared only in terms of precision and recall (the number of content blocks remains the same, but number of the noise blocks is much higher for the latter system). As it is shown in Table 3 the additional low quality data added to the second systems significantly hurt its performance. Precision dropped from 80.75% to 74.78% and recall from 79.88% to 66.95%. We can conclude that the precleaning step where the low-quality web pages (naviga-

[8] *Text* blocks mislabeled as *header* and vice versa are counted too.

tional, image-only, etc.) are removed completely, should be improved.

The speed of the cleaning tool is about 3.5 pages (80 kB) per second. Approximate time for cleaning entire web data set we have (30 million web pages) is 10 days on 10 common CPU cores. We didn't perform this task yet, though.

6. Conclusion and Further Work

We presented a complete solution for cleaning the web pages content, including annotation tool and evaluation metrics. However, this is still an ongoing work and we will continue in research of this challenging task. Current versions of our tools are available for download at `http://ufal.mff.cuni.cz/victor/`.

In the near future, we would like to focus also on real applications – to compare traditional corpora with our web-based corpus using a task that requires large textual data. For Czech, we propose an experiment to compare performance of Czech part-of-speech tagger trained using unsupervised training (Spoustová, 2008) on the data obtained from the cleaned web pages and data from the Czech National Corpus (Institute of Czech National Corpus, 2005).

Acknowledgments

This work has been supported by the Ministry of Education of the Czech Republic, project MSM 0021620838 and the Czech Science Foundation, project 201/05/H014.

We would like to thank Dr. Leo Galamboš for allowing us to obtain the web data and our anonyMouse volunteer annotators for their effort.

7. References

Ziv Bar-Yossef and Sridhar Rajagopalan. 2002. Template detection via data mining and its applications. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*.

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, and Stefan Evert. 2007. Fiasco: Filtering the internet by automatic subtree classification. In *Proceedings of the Web as Corpus Workshop (WAC3), Cleaneval Session*, Louvain-la-Neuve, Belgium.

K. Beesley. 1988. Language identifier: A computer program for automatic natural-language identification on on-line text.

William B. Cavnar and John M. Trenkle. 1994. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, US.

Liang Chen, Shaozhi Ye, and Xing Li. 2006. Template detection for large scale search engines. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, Dijon, France.

Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. 1999. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*.

Leo Galamboš. 2006. Egothor, full-featured text search engine written entirely in java. http://www.egothor.org/.

Katja Hofmann and Wouter Weerkamp. 2007. Web corpus cleaning using content and structure. In *Proceedings of the Web as Corpus Workshop (WAC3), Cleaneval Session*, Louvain-la-Neuve, Belgium.

Institute of Czech National Corpus. 2005. Český národní korpus – SYN2005. http://ucnk.ff.cuni.cz/.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, USA.

Mong-Li Lee, Tok Wang Ling, and Wai Lup Low. 2000. Intelliclean: a knowledge-based intelligent data cleaner. In *Knowledge Discovery and Data Mining*.

Shian-Hua Lin and Jan-Ming Ho. 2002. Discovering informative content blocks from web documents. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton, Alberta, Canada.

Michal Marek, Pavel Pecina, and Miroslav Spousta. 2007. Web page cleaning with conditional random fields. In *Proceedings of the Web as Corpus Workshop (WAC3), Cleaneval Session*, Louvain-la-Neuve, Belgium.

Drahomíra "johanka" Spoustová. 2008. Combining statistical and rule-based approaches to morphological tagging of czech texts. *To appear in Prague Bulletin of Mathematical Linguistics*, 89.

Lan Yi and Bing Liu. 2003. Web page cleaning for web mining through feature weighting. In *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico.

Lan Yi, Bing Liu, and Xiaoli Li. 2003. Eliminating noisy information in web pages for data mining. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, Washington, DC, USA.