Kimatu, a tool for cleaning non-content text parts from HTML docs

Xabier Saralegi[1],Igor Leturia[2]
Elhuyar Foundation

Abstract

This paper explains the functionality of Kimatu, a tool to extract authentic content text from HTML docs –a necessary task to remove linguistically uninteresting text parts–. The system's algorithm consists of a bootstrapping process based in several heuristics and formed by several steps. First, it identifies text blocks that have the same appearance, and calculate for each block a ratio that linearly combines various other ratios useful to measure content-richness. Then it uses blocks with high ratios as references and it sequentially applies other heuristics in order to detect the rest of the candidates and reject some clearly non-relevant blocks.

Keywords: boilerplate removal

1. Introduction

The Internet has become a useful information source for several tasks in NLP and IR fields. However, the textual information present in the Internet –in the form of HTML documents– is usually not structured properly in the sense that content is not semantically tagged. This makes the detection of "boilerplate" elements out of web documents an essential task in any system that uses the Internet as a document information source. The presence of boilerplate disfigures the real content of the document and that can distort linguistic and information extraction processes.

In our case we have developed two tools that use the Internet as a source of documents: Dokusare and AutoCorpEx. Dokusare is a tool for cross-lingual document similarity which links documents on science and technology from different web sites in different languages (Saralegi and Alegria, 2007). AutoCorpEx is a tool that aims at using the Internet to collect corpora in Basque. The presence of boilerplate distorts the representation of the documents and in consequence the performance of similarity measures. We are also working in a collector of comparable corpora from the Internet where the cleaning of documents is clearly necessary in order to obtain a content-text-

[1] Elhuyar Foundation, xabiers@elhuyar.com

[2] Elhuyar Foundation, igor@elhuyar.com

only corpus. So, cleaning boilerplate from HTML documents is a strategic issue in our projects.

2. The tool

The tool we have developed takes a XHTML file as input and returns a text file containing only the real content text with paragraphs, list items and headers identified. This means that a previous HTML to XHTML conversion is necessary. This converter is a script based in HTML Tidy. In some cases it is not possible to perform this previous task perfectly due to the existence of bad formed HTML files. Therefore, sometimes we can lose part of the text in this previous step.

After the conversion to XHTML a program written in Perl applies the algorithm to clean and extract the content text. We use the XML::XPath package to parse the XHTML documents.

3. The algorithm

3.1 The main idea

Our hypothesis is that as HTML is a mainly presentational and visual format, we can use some visual characteristics in order to detect the parts of the document that make up the content. In fact, the design of the document is done to distinguish parts of the document with different functions. That is why in most cases the reader can detect the different parts although he might not understand the language the page is written in. In other cases the HTML text elements are even labeled according to their function in the document. So both kinds of information can be useful to detect the function of each text part.

One of the parts most easily recognizable is the main content. It is normally the biggest part and keeps and homogeneous aspect. Moreover, unlike navigational text, main content tends to have a low density of links and long text elements. Another characteristic is the presence of paragraphs. This can be detected in some languages by means of punctuation marks. Thus, we have designed some ratios that are linearly combined to measure the mentioned characteristics. After applying them, we take text blocks with high ratios as references in order to process other heuristics and complete the output text, because in some cases only the ratios are not determinant. Headers, for example, are not easily captured by means of this ratios and the algorithm must use detected content texts in a bootstrapping way.

3.2 Block detection

The ratios and the next steps of the algorithm are applied to blocks. Blocks are detected before applying the ratios, and are sets of HTML text elements which have the same aspect –or have the same *class* attribute– and are not very distant from each

other. According to our hypothesis all the text in the same block has the same function. So we divide all the document in functional text blocks. The information we use to define the aspect are all attributes and tag names related to the aspect –or class– of the text. Due to the proximity requirement mentioned, there can be different blocks with the same aspect or HTML class.

3.3 The ratios

The way we have computed the heuristics mentioned in the previous section is by means of ratios. We have designed 4 ratios. Two of them are related to the length of the textual content, another one is related to the amount of punctuation signs, and the last one is related to the amount of links. The size ratios are normalized with respect to the document, and the other two are general.

- Length ratio: This ratio measures the total length of the block with regard to the size of the biggest block detected.
- Average length ratio: This ratio measures the average length of the text elements of the block with regard to the highest average length of a block. In order to avoid special extreme cases we take a previously fixed value as a possible maximum for the highest average element size.
- Punctuation sign ratio: This ratio measures the amount of punctuation sings by word in a block with regard to a constant value previously determined manually. We take into account points, quotes... that are used only as phrase boundaries (not contained in hours, acronyms…).
- Link density ratio: This ratio measures the amount of link tags by element in a block with regard to a constant previously determined manually.

The linear combination of the first two heuristics gives us the ratio for being a content text candidate. The weighting is necessary to adjust the ratios. This weighting can be estimated by an exhaustive search, but due to the long time of computation needed, we have only had time to try some of the combinations. The first two ratios are weighted equally and the last two are used only to punish.

Content_relevance_ratio=0.5*(length_ratio)+0.5*(average_length_ratio)-(1-punctuation_sign_ratio)-(link_density_ratio)

3.4 The steps

We have checked that the implemented ratio is useful to detect evident content text –those which are relatively big, with long paragraphs, with punctuations marks and not many links–. However, other blocks that can also be considered content blocks might not have these features. Our strategy is basically to detect all content blocks in a bootstrapping way. Firstly, we detect sure content blocks using the ratio, and then we apply other heuristics that use these sure candidates as references.

This is the algorithm to detect all content texts, which is applied in a greedy way:

- Detect blocks: HTML text elements which have same style and are relatively near.
- Filter by ratio: Assign a content-relevance ratio to each block, and detect sure candidates (ratio>0.32) which will be used as references in the next steps.
- Delete function blocks: Blocks with short elements that share a high percentage of words are rejected. It is very useful to clean informative texts used in forums, e.g. “posted by X”.
- Delete quotes: Taking sure blocks as references, we calculate the length LCS (Longest-Common-Subsequence) ratio between each of these blocks and each of the others if the other appears after the reference ones and have a minimum size. If the LCS ratio is high for most of the blocks of a class all of its candidates are taken as quotes. This is very useful to clean quotes used in forums. Anyway, the quote must be detected as a different block and class from sure blocks.
- Detect dispersed blocks with same style as candidate blocks: When the content elements are disaggregated, some of them can obtain low ratios. In order to rescue them we analyze the blocks of the same class as the reference blocks, and accept those that are near and have a minimum ratio with respect to the reference blocks of the same class (>0.1).
- Delete dispersed candidates: We calculate the geometric mean distance between each candidate and its nearest candidate. This gives a mean distance free of influence from extreme cases. Blocks that have a distance far from the geometric mean and have not a high ratio (>0.5) will be deleted. They are usually texts not related with the content.
- Detect relevant blocks near candidate blocks: Blocks near reference blocks with a minimum global ratio are taken as new candidates. Useful to detect short texts like headers.

Finally, HTML source tags, punctuation signs and size of each text element of candidate blocks are used to select the output tag (h for headers, p for paragraphs and l for list items) of the text element.

4. Other approximations

There are a few tools for cleaning HTML documents. Some are based on tag density, and others on supervised categorization (Lee, Kan and Lai, 2004). The problem of the supervised methods is that the task may be too heterogeneous. The tag density follows an idea similar to our tool’s and bases the cleaning in the detection of extended regions of text (Bernardini et al, 2006). Anyway, it doesn’t take into account some special cases that we treat in our algorithm.

5. Conclusion and future works

Although defining what is non textual information is not a trivial task, we have based our approach in some minimum criteria. Anyway, this subjectivity makes it difficult to design algorithms and evaluating the results precisely.

HTML documents are very heterogeneous, and some of them which are very difficult to clean (some kinds of forms, short documents, texts with very anarchic structure...) are often not very interesting for NLP tasks. We think that trying to clean all documents can be a very ambitious objective. Maybe it would be better to focus our efforts in trying to clean only those that meet some minimum requirements. Moreover, we doubt whether cleaning will be necessary in the near future. The newest tagging standards (RSS, RDF, microformats...) cover more kind of information: aspect, content structure, semantics..., so cleaning might become easier.

Anyway, we plan to improve our algorithm in some aspects. Apart from the mentioned ratios we plan to add one based on querying APIs of search engines in order to detect frequent blocks among different pages of the same site. Another ratio that we have in mind, but that we have not had time to try either, is based in distribution of different POS tags. As for the algorithm we plan to model it as a backtracking algorithm.

Bibliography

Lee C. H., Kan M., and Lai S. (2004), *Stylistic and lexical cotraining for web block classification*, in WIDM 2004. 136143

Bernardini S., Baroni M. And Evert S. (2006), *A WaCky introduction*, in Working papers on the Web as Corpus, Bologna: Gedit. 9-40.

Saralegi X. and Alegria I. (2007), *Similitud entre documentos multilingües de carácter científicotécnico en un entorno Web*, in SEPLN 2007.