Web Article Extraction for Web Printing: a DOM+Visual based Approach

Ping Luo, Jian Fan, Sam Liu, Fen Lin, Yuhong Xiong, Jerry Liu
Hewlett Packard Labs
{ping.luo, jian.fan, sam.liu, fen.lin, yuhong.xiong, jerry.liu}@hp.com

ABSTRACT

This work studies the problem of extracting articles from Web pages for better printing. Different from existing approaches of article extraction, Web printing poses several unique requirements: 1) Identifying just the boundary surrounding the text-body is not the ideal solution for article extraction. It is highly desirable to filter out some uninformative links and advertisements within this boundary. 2) It is necessary to identify paragraphs, which may not be readily separated as DOM nodes, for the purpose of better layout of the article. 3) Its performance should be independent of content domains, written languages, and Web page templates. Toward these goals we propose a novel method of article extraction using both DOM (Document Object Model) and visual features. The main components of our method include: 1) a text segment/paragraph identification algorithm based on line-breaking features, 2) a global optimization method, Maximum Scoring Subsequence, based on text segments for identifying the boundary of the article body, 3) an outlier elimination step based on left or right alignment of text segments with the article body. Our experiments showed the proposed method is effective in terms of precision and recall at the level of text segments.

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation—*hypertext/hypermedia, languages and systems, markup languages, scripting languages*; I.2.6 [**Artificial Intelligence**]: Learning—*concept learning*

General Terms

Algorithms, Experimentation

Keywords

Article extraction, maximal scoring subsequence

1. INTRODUCTION

Many people find the seemly simple task of printing web pages often frustrating because of common problems such as too many pages generated, poor page layout, and distracting ads at conspicuous locations - just to name some. The problem mainly stemmed from how information is represented on a web page, which is encoded in a language (HTML) that is not targeted for printing. HTML can be messy and confusing to interpret by the current browser print utility, so the web page that is rendered by the browser does not match the print output. As an illustration of current print problems, we randomly select a recent Web page[1] in Figure 1(left) from SL.com, a prime candidate for printing news worthy articles. As one can try, the printout provided by the Web browser suffers from including a large area of un-informative content (navigation menu, ads and related links) in it. More importantly, this poor user experience would surely discourage future printing from the web. To improve web printing, we like to have the much preferred printout in Figure 1(right), which contains only the title, article text-body, and associated images.

Figure 1: An example Web article with bounding rectangles of text segments (left), our Printout of the extracted article text, title and image (right).

Providing friendly user experience for Web printing is strongly dependent on accurately extracting desired information from semi-structured HTML documents, which has been well-studied in previous work. Here, we only discuss the related work of two recent papers [1, 3], which are the state of the art in article extraction. Wang et al. [3] propose a template-independent method for Web news extraction.

[1]http://sportsillustrated.cnn.com/2009/soccer/05/27/champions.league.ap/index.html

DocEng'09, September 16–18, 2009, Munich, Germany.

Based on some visual features this method learns the wrapper to identify the minimal DOM sub-tree which contains the whole article text-body. Pasternack and Roth [1] introduce a global heuristic of maximum subsequence segmentation based on word-level local classifiers, and apply it to the domain of news Web sites. However, these methods are ill suited for Web printing for the following reasons:

• Both of these methods only identify the boundary of the text-body. Since some unwanted content such as link-lists (to related stories) and ads, may exist within this block, article extraction at this level is not the ideal solution for Web page printing. Pasternack and Roth [1] argue that removing whatever junk remains becomes much simpler, and also propose some simple rules to do it. One of these rules is to remove the contents of any *div* HTML tag pair in the DOM that contains a hyperlink. However, it is quite often that the article paragraphs include some links within them. Thus, this rule would incorrectly remove these paragraphs. Additionally, by our experiences of reading hundreds of Web articles it is not a trivial task to remove noises within the text-body.

• Both of these methods cannot detect the paragraph separation within the text-body. For a better printable article we need this information for the re-layout of text-body with title and image. Due to the heterogeneity of HTML tags used to separate paragraphs it is not a trivial task neither.

• The method from Pasternack and Roth [1] is dependent of content domains and writing languages. That is to say, the model trained on the data in one content domain (or language) is not applicable to the data in another domain (or language). For example, the model trained based on news articles is not applicable to healthcare articles. The reason is that this model uses the word-level features in the local classifier. However, as a general solution of article extraction for Web printing the method must be independent of content domains, writing languages, and Web page templates.

In this paper we propose the method of article extraction which addresses the above challenges. First, based on line-breaking features we present a Web page segmentation algorithm to generate *text segments*, the basic elements for processing, each of which may correspond to a paragraph. We then use a global optimization method, *Maximum Scoring Subsequence*, over segment-level (not word-level or DOM tree leaf node level) local classifier, and apply it to detect the boundary of text-body. Finally, we propose a novel method of left and right alignment to remove the unwanted content in the text-body block. Since we only use the DOM and visual information in our method, it is independent of content domains, writing languages, and Web page templates.

2. PROBLEM DESCRIPTION

In this work we adopt the same description of *article* in [1]. *By "article" we mean a contiguous, coherent work of prose on a single topic or multiple closely related topics that alone comprises the main informational content of the page – news stories, encyclopedia entries, or a single blog post are considered articles, whereas a collection of headlines with brief summaries, a list of search results, or a set of blog posts are not.*

While a Web article is unstructured in HTML/DOM, it does have some sense of visual structure. For example, the article body usually consists of contiguous paragraph blocks occupying the main area of the Web page. As for the article title it usually has more standout font and always positioned

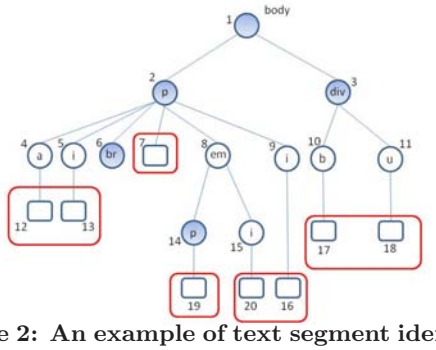

Figure 2: An example of text segment identification. Nodes with solid blue color are line-break nodes. Red round rectangles indicate text segments.

before the text-body. For a Web article, we extract the title, text-body, and non-ads image for Web printing. This task is complicated by the presence of less informative and typically unrelated material, such as links to related stories, navigation menu, and ads. In this paper we mainly describe our work on article text-body extraction.

3. IDENTIFICATION OF TEXT SEGMENTS

The first task is to define the basic processing element for article extraction. Trivially, we can consider the *text DOM leaf nodes* (*text leaf nodes* for short) as the basic elements. However, this element might be too primitive for article extraction. For example, it is quite often that there exist several links in a paragraph of a news article. Thus, the leaf nodes within the link HTML tags will divide the whole paragraph into multiple pieces. However, ideally we want the whole paragraph intact and to be the basic processing element (its advantage will be discussed later). In this section we will give the definition of *text segment* that meets this requirement.

3.1 Identification of Line-break Nodes

After analyzing many HTML source files we find that a paragraph in a Web article can be created using different tags, including some paired-tags such as *div*, *p*, and *blockquote*, or some single-tags such as *br* and *hr*. Although the tag used to form a paragraph is different it is always true that there does not exist any line-break within a paragraph. Thus, generally speaking, we want to group the successive text leaf nodes between two line-breaks into a text segment.

looks at css display property

Now the problem is how to get the line-break information from the DOM tree and the display information of the Web page. In our work we consider the following two rules to obtain the line-breaks among text. 1) The CSS display property. The CSS display property sets how/if an element is displayed. If the display property of a node is "block", this element will be displayed as a block-level element, with a line-break before and after the element. 2) The single (not paired) tags *br* and *hr* will generate a line-break after them. The first rule considers the line-breaks before and after some paired-tags tags, such as *div*, *p*, and *blockquote*. Since it is hard to list all such tags that generate line-breaks we leverage the CSS display property on each tag. Note that this information can only be obtained after rendering this page. Furthermore, the second rule considers the line-breaks after some single-tags, such as *br* and *hr*. This can be obtained through parsing the DOM tree.

For convenience in description we give the definition of *line-break nodes* as follows.

Table 1: The leaf nodes and their NLNs

Text Leaf Node	NLN	Text Leaf Node	NLN
n_{12}	n_2	n_{13}	n_2
n_6	n_6	n_7	n_2
n_{19}	n_{14}	n_{20}	n_2
n_{16}	n_2	n_{17}	n_3
n_{18}	n_3		

looking for some of text segments

DEFINITION 1 (LINE-BREAK NODE). *A DOM node is a line-break node if and only if the value of its display property is "block" or the HTML tag of this node is* br *or* hr.

We give an example DOM tree in Figure 2 to clarify how to generate text segments. In this figure The number beside each node is its index number. The circular nodes are the HTML tag nodes, while the rectangular nodes are the text leaf nodes. (Note that some HTML tag nodes can also be a leaf node in the DOM tree. For example, Node 6 in Figure 2 is a leaf node with the single-tag *br*.) The circular nodes with solid blue color are the line-break nodes. After rendering this DOM tree the corresponding Web page appears as follows:

text in n_{12} *and* n_{13}
text in n_7
text in n_{19}
text in n_{20} *and* n_{16}
text in n_{17} *and* n_{18}

where n_i is the i-th node in the DOM tree. Ideally we would like to get the following five text segments: $\{n_{12}, n_{13}\}$, $\{n_7\}$, $\{n_{19}\}$, $\{n_{20}, n_{16}\}$, $\{n_{17}, n_{18}\}$, each of which corresponds to a line in the Web page.

It is quite common that the sub-tree of a line-break node contains another line-break node. For example, in Figure 2 the sub-tree of line-break Node 2 contains two other line-break nodes, Node 6 and Node 14. Due to this nesting relationship among line-break nodes, it is not trivial to group the text leaf nodes into a text segment.

3.2 Bottom-up Grouping of Text Nodes

To group the text leaf nodes, we first identify the *Nearest Line-break Node* (NLN), defined in Definition 2, for each leaf node on the DOM tree.

DEFINITION 2 (NEAREST LINE-BREAK NODE). *Given a leaf node n in a DOM tree T, the nearest line-break node for n is the first line-break node along the shortest path from n to the root of T.*

Note that in the above definition the shortest path includes the two end points, node n and the root of T. It indicates that if a leaf node is a line-break node (eg. Node 6 in Figure 2), its NLN is itself. Table 1 lists all the leaf nodes and their NLNs in the example tree in Figure 2.

By the following steps we can group the text leaf nodes into multiple text segments.

1. Generate the sequence of all the leaf nodes, which are ranked in the order that they appear in the HTML file. For the example in Figure 2, this sequence is $(n_{12}, n_{13}, n_6, n_7, n_{19}, n_{20}, n_{16}, n_{17}, n_{18})$. Note that in this list n_6 is a single-tag node, not a text leaf node.

2. Group the successive nodes in this sequence with the same NLN into a text group. For the example in Figure 2 we get 6 groups: $g_1 = \{n_{12}, n_{13}\}, g_2 = \{n_6\}, g_3 = \{n_7\}, g_4 = \{n_{19}\}, g_5 = \{n_{20}, n_{16}\}, g_6 = \{n_{17}, n_{18}\}$. In this example, n_7 and n_{20} has the same NLN of n_2, but they are not successive to each other in the sequence of leaf nodes. Thus, these two nodes are not in a group.

3. For each group generated by Step 2, we check whether it only contains a single-tag node. If it does, remove this group. In our example $g_2 = \{n_6\}$ is such a group. Thus, it is removed from the final result.

DEFINITION 3 (TEXT SEGMENT). *Each group of text leaf nodes, generated from the above process, is a text segment.*

Finally, we get the five text segments: $\{n_{12}, n_{13}\}$, $\{n_7\}$, $\{n_{19}\}$, $\{n_{20}, n_{16}\}$, $\{n_{17}, n_{18}\}$, which is the correct result.

4. ARTICLE EXTRACTION

Given a sequence of text segments $\vec{s} = (s_1, \cdots, s_n)$ (where s_i is a text segment) generated from a Web page by the algorithm in Section 3.2, we aim to identify the subsequence of $\vec{s}$ which exactly bounds the text-body.

4.1 Maximum Scoring Subsequence of Text Segments

Providing a sequence $\vec{v} = (v_1, \cdots, v_n)$, where $v_i \in \mathcal{R}$, a Maximum Scoring Subsequence (MSS) of $\vec{v}$ is a sequence $T = (v_a, v_{a+1}, \cdots, v_b)$ where $1 \le a \le b \le n$ and a and b are given by

$$(a, b) = \arg\max_{x,y} \sum_{i=x}^{y} v_i \quad (1)$$

The complexity of the algorithm for the above problem is linear with respect to the length of $\vec{v}$ [2].

Typically, the text-body, a contiguous block of text with common font size (and color) and without too many links and decoration tags, always occupies the main area of the Web page. Thus, we adopt the algorithm of MSS to identify the range of the text-body. Specifically, we use a segment-level classifiers F (a function from a text segment to a value in $[-1, 1]$) to give a score for each segment. The bigger this value is, the more probable that the segment is in the text-body. We then solve the following MSS problem.

Given a sequence of text segments $\vec{s} = (s_1, \cdots, s_n)$ (s_i is a text segment), we solve the MSS problem in Equation (1) for the value sequence $\vec{v} = (v_1, \cdots, v_n)$, where

$$v_i = F(s_i) \cdot StringLength(s_i), \quad (2)$$

$StringLength(s_i)$ is the string length of the text in s_i. The resultant text segments in this maximum scoring subsequence gives the bounding block of the text-body.

The local classifier F gives any text segment a value, which determines whether it is in the text-body. The text-body is likely to satisfy the following specifications: 1) the font size (color) used in any paragraph of the text-body may be the most commonly used one in the Web page; 2) There may not be too many links in a paragraph of the text-body.

Thus, we can generate the format-based features for the local classifier of the text segment. Specifically, for each text segment we compute the percentage (in terms of string length) of the text in this segment that appears in a certain format. The formats we consider in this work include: 1) whether it is in the most commonly-used font size; 2) whether it is in the most commonly-used font color; 3) whether it is the text in a link. Then, these three format requirements correspond to the three values of percentage, denoted by p_{size}, p_{color} and p_{link}, respectively. For a given Web page the commonly-used font size and color can be calculated in advance in terms of string length. We can train this local classifier based on the values of p_{size}, p_{color} and p_{link} on each text segment. However, in the current work we manually formulate this function of F in the following form,

$$F(s) = \begin{cases} 1 & p_{size} \ge c_1, p_{color} \ge c_2, p_{link} \le c_3 \\ -1 & \text{otherwise,} \end{cases} \quad (3)$$

where c_1, c_2 and c_3 are three constant values, which are heuristically set to 0.7, 0.2, and 0.5 respectively. Later, the experimental results will show the effectiveness of this simple function in article extraction.

4.2 Text-body Refinement by Horizontal Alignment

After identifying the range of the text-body by MSS we leverage the horizontal alignment property to further remove the junk in this range. The heuristic rule of horizontal alignment is based on the fact that the article paragraphs often overlap largely in horizontal direction.

After rendering a Web page the Web browser can provide the position information for each HTML tag node in the DOM tree. This position information of a tag node is expressed as the rectangle, which bounds the area of the node and its descendants. Since all the text leaf nodes in a text segment share the same NLN (see the details of the generation of text segments in Section 3) we define the *bounding rectangle of any text segment* as the rectangle on the shared NLN of the text leaf nodes in this segment. As shown in Figure 1(left), the bounding rectangles of all the text segments in the identified range of article body are also depicted. It is clear that the bounding rectangles (with solid line) of the text segments for the article paragraphs have the same left and right coordinates. However, the bounding rectangles (with dash line) of the text segments for the caption below the image and the links to the related stories (which are the junk within the text-body boundary) only cover a small part of the horizontal range of the article paragraphs. Thus, we use this property to further remove the unwanted content in the bounding block of the article body.

5. EXPERIMENTAL EVALUATION

Some initial experiments have been conducted to test the effectiveness of this method of Web article extraction. We use the test data set of 95 news pages crawled from 10 online news sites, such as CNN, ESPN, BBC, NYTimes and TechCrunch. We manually label all the text segments in the text-body on each of these pages, and compare the truth with the prediction. The evaluation metrics include the precision and recall, which are calculated at the level of text segments. If we take S_p as the set of text segments in the extracted results, and S_l as the set of text segments in the labeled results, then precision and recall are given by

$$P = \frac{|S_p \cap S_l|}{|S_p|}, R = \frac{|S_p \cap S_l|}{|S_l|}. \quad (4)$$

The experimental results with a precision of 91.571% and a recall of 99.145% show that this method is quite effective in text-body extraction.

It is worth mentioning that the evaluation metrics, the precision and recall at the level of words, used in the recent paper [1] are totally different from the ones defined here. Specifically, they calculate

$$P' = \frac{|W_p \cap W_l|}{|W_p|}, R' = \frac{|W_p \cap W_l|}{|W_l|}, \quad (5)$$

where W_p and W_l are the set of words in the predicted and labeled extraction, respectively. Since the words in the text-body are likely to be much more than the ones outside the text-body, these two values of P' and R' tend to be high. Thus, we argue that the evaluation based on P and R are more strict than that based on P' and R'.

We have compared our method with the one in [1] on some examples of Web articles. They provide an online demo[2] for the supervised version of their method. We tested this demo on some recent news articles from CNN and found that the extracted results from our method are much better. For example, on these pages their method also output the caption below the article image and the links to the related stories as the result. However, our method outputs the exact text-body. Additionally, their result also contains the HMTL tags and does not identify the separations among paragraphs, which make it hard to read. We must mention that their method is faster than ours since they do not use the visual information which is obtained only through the time-consuming rendering of the Web page. The systematic comparison between these two methods will be our future work.

We have combined our method of article extraction and the Tabblo Print Toolkit[3] to generate a readable and printable PDF file for a given Web article. Specifically, we feed the extracted results, including the text in each paragraph, the title and image, to TPT. Then, TPT uses the article template to re-layout the whole page. Figure 1(right) is the actually the PDF file, generated by our method for the Web page in Figure 1(left). Some testing on Chinese Web articles also shows that this method is independent of writing languages.

6. CONCLUSIONS

In this paper we propose an effective approach of Web article extraction, which addresses several new challenges imposed by Web page printing. In preprocessing the Web page we leverage the line-break features to generate text segments, each of which may correspond to a paragraph in the text-body. And then, based on the local features on each segment and the global optimization of MSS over the sequence of segments we identify the range of the text-body. Finally we propose the heuristic rule of left and right alignment to remove the junk in the text-body. Our contributions include the generation of text segments, the segment-level local classifier with some novel features, and the left and right alignment for text-body refinement. Our initial experiments show very high success of this method even when the segment-level local classifier is manually formulated to a simple form. We believe that only deep semantic analysis in text and image can further improve the performance of article extraction.

7. ACKNOWLEDGMENTS

The authors would like to thank Demiao Lin for his work on developing the labeling tool for Web news metadata.

8. REFERENCES

[1] J. Pasternack and D. Roth. Extracting article text from the web with maximum subsequence segmentation. In *Proceedings of the 18th WWW*, 2009.

[2] W. Ruzzo and M. Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *Proceedings of ISMB*, 1999.

[3] J. Wang, X. He, C. Wang, J. Pei, J. Bu, C. Chen, Z. Guan, and W. V. Zhang. Can we learn a template-independent wrapper for news article extraction from a single training site? In *Proceedings of the 15th SIGKDD*, 2009.

[2] http://took.cs.uiuc.edu/MSS/

[3] http://www.tabblo.com/