Automatic Web Content Extraction by Combination of Learning and Grouping

Shanchan Wu, Jerry Liu, Jian Fan
HP Labs
1501 Page Mill Road, Palo Alto, CA, 94304
{shanchan.wu, jerry.liu, jian.fan}@hp.com

ABSTRACT

Web pages consist of not only actual content, but also other elements such as branding banners, navigational elements, advertisements, copyright etc. This noisy content is typically not related to the main subjects of the webpages. Identifying the part of actual content, or clipping web pages, has many applications, such as high quality web printing, e-reading on mobile devices and data mining. Although there are many existing methods attempting to address this task, most of them can either work only on certain types of Web pages, e.g. article pages, or has to develop different models for different websites. We formulate the actual content identifying problem as a DOM tree node selection problem. We develop multiple features by utilizing the DOM tree node properties to train a machine learning model. Then candidate nodes are selected based on the learning model. Based on the observation that the actual content is usually located in a spatially continuous block, we develop a grouping technology to further filter out noisy data and pick missing data for the candidate nodes. We conduct extensive experiments on a real dataset and demonstrate our solution has high quality outputs and outperforms several baseline methods.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models, Information filtering*

General Terms

Algorithms, Experimentation, Performance

Keywords

Content Extraction; Information Extraction; Web Page Segmentation; Noise Removal; Information Retrieval; Web Mining

1. INTRODUCTION

As more and more information is published on the World Wide Web, the influence of the World Wide Web increases. Thus, discovering knowledge from web and providing users with useful information have become important and popular. For this purpose, main content in a web page must be identified. This is a challenging task, since the main content in a web page is often accompanied by a lot of additional and often distracting content such as branding banners, navigation elements, advertisements and copyright etc, and the web pages in the World Wide Web are highly heterogeneous. To extract useful knowledge from a web page and to provide users clean content to read, this additional information should be treated as noise and removed from the main content.

The task of extracting the main content from web pages has attracted many research works [4, 10, 12, 13, 18]. However, most of these previous works focus on article web pages, such as news articles, and blog posts. These article pages show some similar layout patterns, although the styles are usually different from website to website. Thus some heuristics from the visual features can be used to extract these informative items with pretty high accuracy [4].

As article news pages are likely be are generated from some underlying structured sources, some template-based solutions make the assumption that there exists a template structure in news pages for a specific news website. Some template based approaches such as Tree Edit Distance (TED) exploit the similarities of the structures in HTML pages and generate wrappers [14]. However, the generated wrappers can only work properly for news pages which are constructed on a template that has been seen in the training data. Furthermore, any small changes in the underlying HTML structures will be very likely to invalidate the wrappers, and it is computationally and resource expensive to maintain up-to-date wrappers for the huge amount of news websites.

Besides article pages, there are also a large volume of non-article web pages, such as online shopping pages and recipe pages. The layout patterns of the non-article web pages are more difficult to be generalized. These previous solutions that were only demonstrated to work for article web pages are not very likely to work for non-article web pages. To help users to get more accurate main content, the tool of Smart Print provides for Web page printing [10] provides users an option to manually select the informative areas of a web page in an interactive interface. The more manual work that needs to get involved in the tool, the worse the user experience would be. A good extraction solution that works well for all kinds of web pages will reduce manual work and hence would help to improve user experience.

There are some previous learning approaches to classify web page regions as actual content regions and noisy regions based on some features extracted from the regions [18]. Usually the output for a region with either "YES" or "NO" does not take the surrounding areas into consideration, i.e. an isolated binary decision. A region by itself may look like or be classified as a noisy or non-noisy region. However, after taking the surrounding areas into consideration, a region which was classified as a non-noisy region may

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741659.

turn out to be a noisy region, and vice versa. Furthermore, most of the previous approaches are focused on article web pages, which are relatively simpler to extract actual content from compared with general webpages. In general, web pages may contain tables, figures or other visual formats for actual content in addition to text. Hence, regions of general web pages are more likely to be misclassified than those of article web pages. We expect that incorporating some techniques that take the surrounding areas into consideration would help to improve the classification accuracy.

We propose a novel approach for automatic extraction of web content in a webpage by combination of learning and grouping. We formulate the problem as a DOM tree node selection problem. To train a learning model, we develop multiple features by utilizing the DOM tree node properties. We consider the features of positions, areas, fonts, text, tags and links. We calculate and adjust the feature values into the formats more suitable for the learning model. After the learning model is trained, it is used to score the nodes and select the candidate nodes. We compact the candidate nodes by removing the redundant nodes. With the observation that the actual content is usually located in a spatially continuous block, we develop a grouping technology to further filter out noisy data and pick missing data for the candidate nodes. We conduct extensive experiments on a real dataset and demonstrate our solution has high quality outputs and outperforms several baseline methods.

2. RELATED WORK

Extracting web content from web pages has received substantial interest by researchers [13, 4, 6, 10, 12, 18, 21, 8, 17, 9, 19, 16]. Most of existent working has focused on article web pages. For example, Pasternack et. al. [13] proposed a method based on utilizing maximum subsequence segmentation to extract the text of articles from HTML documents using features of tri-grams and tags. Wang et al. [18] proposed an approach by learning a template-independent wrapper for news article extraction to identify one minimum sub-tree containing title and and one minimum sub-tree containing article body content. Luo et al. [12] utilized line-break features to generate text segments corresponding to paragraphs and then find the range of text body by the maximum subsequence algorithm. They also proposed some heuristic rules to filter out the noisy data within the text body. Fan et al. [4] proposed a system to extract titles, main text body, content- related images and image captions from news- and blog-like article web pages. Kohlschutter et al. [8] analyzed shallow text features - number of words and link density - to distinguish main content from other parts of information from news article web pages.

Weninger et al. [19] proposed the Content Extraction via Tag Ratios (CETR) method to extract content text from diverse web pages using HTML tag ratios. They computed the tag ratio for each line and then clustered the resulting histogram into content and noise areas. Sun et al. [17] proposed Content Extraction via Text Density (CETD) to extract content from web pages, based on the observation that the content is usually containing more text and simply formatted, while noise is usually highly formatted and contains less text with short sentences.

One approach for content extraction is the Template Detection (TD) algorithms [1, 3, 7, 11, 20] which applies collections of documents with the same templates to learn common structures. Bar-Yossef et al. presented an approach to automatically detect templates by counting the frequent pagelet item sets, where each pagelet is a self-contained logical regions in a web with a well defined topic or functionality[1]. Lin et al. partitioned a web page into several blocks by HTML tag <table>, and utilized entropy measure to partition blocks into informative or redundant by a set of word based

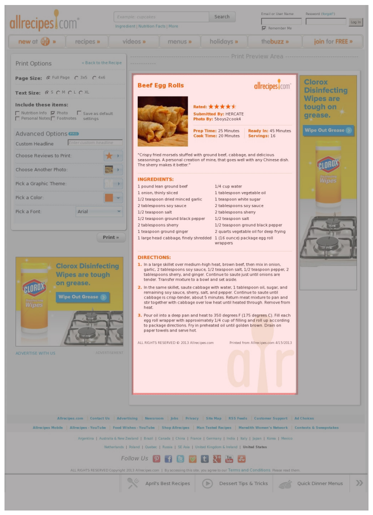

Figure 1: An example of extracting the main content from a webpage. The area with light red background is the main content area.

features [11]. Yi et al. introduced a site style tree (SST) structure to distinguish noisy blocks and main content blocks of the Web site [20], based on the observation that noisy blocks are more likely to share some common contents and presentation styles, while the main content blocks of the pages are more likely to be diverse in their actual contents and/or presentation styles. Chen et al. proposed a two-stage template detection method to combine template detection and removal with the index building process of a search engine. They segmented web pages into blocks, and blocks are clustered according to their style features. The blocks with similar layout style and content were detected and identified as templates [3]. All these template detection algorithms generally try to identify the main content by removing common parts found in different web pages. This is based on the assumption that the noisy parts such as advertisements are more likely to be repeated across different web pages. As different web sites are designed with different styles, in order to more successfully identify the main content different template models need to be built for different web sites. Even each website is provided with a unique template model, it would meet difficulties when different design styles are used in different web pages in the same website. Furthermore, the update of the layout or structure of the pages in the web site will invalidate the template model.

Cai et al. [2] proposed the Vision-based Page Segmentation (VIPS) technique to segment a web page into semantically related content blocks from its visual presentation, based on some heuristic rules on the visual information. They constructed a tree structure for a web page, where the nodes to the tree are visually grouped blocks. Based on the VIPS technique, Song et al. [15] presented an approach to rank block importance for web pages by learning from block features (including spatial features and content features). Fernandes et al. [5] proposed a different method to compute block importance of a web page by assigning weights to classes of structurally similar blocks.

3. PROBLEM FORMULATION AND SOLUTION

The task of extracting main content from a web page is to select a set of informative areas in the web page. Figure 1 shows an ex-

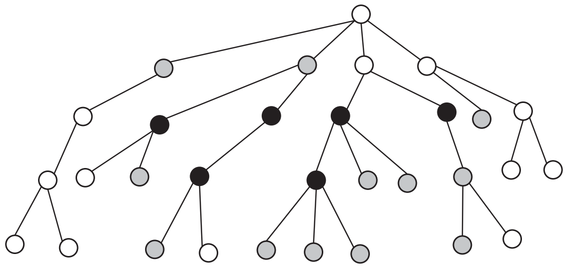

Figure 2: DOM tree.

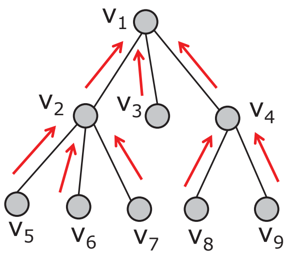

Figure 3: Feature calculation.

ample of extracting the main content from a recipe web page. Web pages are encoded by the HTML language and a rendering engine can render it to a DOM tree structure. Figure 2 shows a simple DOM tree structure. For most of the web pages, the sizes of their DOM trees are usually much bigger than the size of the DOM tree in this example. Each node in the DOM tree can be labeled with a HTML tag, e.g. “TITLE”, “P”, “A”, “HEAD”, and each node is associated with other properties, like position information, font information, text information etc. To simplify the expression, we use a DOM tree node to represent the a block of content of this node and all of its descendant nodes. The task of extracting main content from a web page may be formulated as a problem of node selection from its DOM tree of the web page.

To solve this problem, we propose a solution with two major steps as following.

The first step is the learning step. In this step we train a learning model using the training data. Based on the learning model, we assign a score to each node in the DOM tree of a web page. The scores represent how likely the corresponding nodes are part of the main content, i.e., the content which we recommend to users and interest users. We select the candidate nodes by a threshold.

The second step is the grouping step. As the output from the learning step is not perfect, in this step we group candidate nodes, remove the noisy groups, and refine the selected group by further removing noisy nodes and picking up missing nodes.

We will describe more details about the solution in the following sections.

4. FEATURE SELECTION

In this section, we describe how we select and calculate features for DOM tree nodes that will be used for a learning model.

As we use a node in the DOM tree to represent the regions covered by this node and all its descendants, we set the features of a node to be able to represent the aggregated properties of this node and all its descendants. For a feature, we have three types of values: one is absolute value, one is relative value, and one is adjusted value. The absolute value is the real value of this feature, for example, the area size in square inches. The relative value is the value that is the normalized value of this feature. For example, the relative value of the area size feature of a node is the area size covered by this node and all its descendants divided by the area size of the whole page. The adjusted value is the value that is transformed from the relative value to a format more suitable for a learning model. For some features, we can just set the adjusted value to be the same as the relative value, and use it for the learning model.

For a node v_i, the absolute value of its feature x is recursively set to be the union of its original x property value and the absolute values of the feature x of all its children:

$$F_x(v_i) = F'_x(v_i) \bigcup \left\{ \bigcup_{v_j \in Children(v_i)} F_x(v_j) \right\} \quad (1)$$

In equation (1), $F_x(v_i)$ is the absolute value of feature x for node v_i and $F'_x(v_i)$ is the original x property value of node v_i, and $F_x(v_j)$ is the absolute value of feature x for node v_j. If a node does not have any children, the absolute values of all its features are set to be its corresponding original property values. The property values of a node are directly available after the web page is rendered by a rendering engine, e.g., Webkit. For different features, the union operation in equation (1) is slightly different. For the number of characters, the union operation is mathematical addition. For font color and font size, the union operation is merging the pairs of key and values, e.g., $\{red : 4, green : 2\} \bigcup \{red : 3, blue : 1\} = \{red : 7, green : 2, blue : 1\}$, where $red : 4$ means the number of red characters is 4. For left, right, top and bottom positions, the union operation is to get the corresponding left most or top most or right most or top most or bottom most positions. For area size, the union operation calculates the area size of the union regions, rather than adding them up since there might be some overlap.

Figure 3 shows the way about how the absolute feature values are calculated. In the DOM tree, the absolute feature values are calculated by traversing the tree from the bottom level to the top level, level by level. In Figure 3, the absolute feature values of nodes v_5, v_6, v_7, v_8, v_9 which are in the bottom level are first calculated, and then the absolute feature values of the nodes v_2, v_3, v_4 which are in one upper level are calculated, and at last the feature values of root node v_1 are calculated. After getting all of the absolute feature values, we can get the relative feature values and the adjusted feature values by visiting each node once more.

We are considering position features, area features, font features, text, tag and link features.

4.1 Position and Area Features

For the position features, we consider the left, right, top, bottom, horizontal center and vertical center positions. As we have mentioned earlier, for the left, right, top, bottom positions of a node, we calculate its absolute value using equation (1). Since there are no original horizontal center and vertical center position properties from the rendering output, their absolute values are the mean value of the absolute value of the left position and the absolute value of right position, or the the mean value of the absolute value of the top position and bottom position. We can easily get the relative values of these positions by normalization with the page width or the page height, depending on the types of the positions.

After we get the relative values of the positions, we then calculated the adjust values based on a measure of how good are the positions. For left position, a “perfect” relative left position value is the average value of the relative left positions of all of the ground truth blocks in the training dataset. Suppose this "perfect" value is $BEST\_LEFT$. Then we calculate the adjusted value of left position feature of a node as follows:

$$POS\_LEFT = 1 - |BEST\_LEFT - LEFT|$$

where POS_LEFT is the adjusted value of the left position feature and $LEFT$ is the relative value of the left position. The value is in the range of [0,1] and the higher the better. Similarly we can calculate the adjusted values of left, right, top, bottom, horizontal center and vertical center positions, which we label as POS_LEFT, POS_RIGHT, POS_TOP, POS_BOTTOM, $POS\_CENTERX$, and $POS\_CENTERY$.

We add one feature to capture the distance between the center of a node to the "perfect" center position, which only has the adjusted value. The calculation is based on the relative values of the center positions and we label this adjusted feature value as POS_DIST.

For area features, we consider the area size. We set the adjusted value of area size to be the same as the relative value, i.e., the normalization of the area size of a node, which we label as $AREA\_SIZE$.

We also add one feature to capture how close is the area size to the "perfect" area size, which only has the adjusted value. A "perfect" relative area size value is the average value of the relative area size values of all of the ground truth blocks in the training dataset. If we use the difference between the relative area size value of a node and the "perfect" relative area size value to measure the closeness, then a very big area size will almost always be worse than a very small area size, which is not fair. So we calculate the logarithm value of the relative area size value and the logarithm value of the "perfect" value, and use the difference of these two values to measure the closeness, which is more reasonable. We label this adjusted feature value as $AREA\_DIST$.

4.2 Font Features

For font features, we consider font color and font size. The absolute value of the color feature of node k has a form of a set of pairs of key and value, i.e., $\{c_1 : m_{k1},\ c_2 : m_{k2},\ \cdots,\ c_i : m_{ki},\ \cdots,\}$, where c_i is the color ID and m_{ki} is the number of characters with color c_i in that node. As we have shown previously, this absolute value can be calculated by the union operation from bottom level of the DOM tree to upper level of the DOM tree. level by level. After we have the absolute value, we can calculate the relative value in the similar format of $\{c_1 : \varphi_{k1},\ c_2 : \varphi_{k2},\ \cdots,\ c_i : \varphi_{ki},\ \cdots,\}$, where c_i is the color ID and φ_{ki} is the percentage of characters with color c_i in node k, with $\varphi_{ki} = \frac{m_{ki}}{\sum_j m_{kj}}$.

Based on font color, we introduce a feature to capture how much the colors of the characters in a node matches the majority of the colors of the characters in the whole page. We call this feature as font color popularity, which only has the adjusted value. We note that the relative value of the color feature of the root node r, $\{c_1 : \varphi_{r1},\ c_2 : \varphi_{r2},\ \cdots,\ c_i : \varphi_{ri},\ \cdots,\}$, can also represent distribution of the colors of the characters in the whole page. We calculate the font color popularity value of node k as:

$$FONT\_COLOR\_POPULARITY = \sum_i \varphi_{ki}\varphi_{ri}$$

Similar to font color, for font size, we can calculate the relative value of the font size of node k in the format of $\{z_1 : \rho_{k1},\ z_2 : \rho_{k2},\ \cdots,\ z_i : \rho_{ki},\ \cdots,\}$, where z_i is the font size with size value z_i and ρ_{ki} is the percentage of characters with font size z_i in node k. Suppose the minimum font size in the webpage is z_{min} and maximum font size in the webpage is z_{max}. We calculate the font size adjusted value as:

$$FONT\_SIZE = \sum_i \frac{\rho_{ki}(z_i - z_{min})}{(z_{max} - z_{min})}$$

Similar to font color popularity, for font size, we also have font size popularity to capture how much the character sizes in node k matches the majority of the character sizes in the whole page (represented by root node r), which is calculated as:

$$FONT\_SIZE\_POPULARITY = \sum_i \rho_{ki}\rho_{ri}$$

4.3 Text, Tag and Link Features

For text, we consider the number of visible characters and the text area size. After we get the absolute value of the number of visible characters of a node, its relative value is normalized by the total number of visible characters in the page. We set the adjusted value of the number of visible characters to be the same as the relative value and label it as $VISIBLE\_CHAR$.

We can also calculate the absolute value of text area size and the absolute value of image area size for each node. We want to have a feature to capture whether a node has more text area or more image area. This feature only has adjusted value and is calculated as:

$$TEXT\_RATIO = \frac{A_{text}}{A_{text} + A_{image} + 1}$$

where A_{text} is the text area size of a node (absolute value) and A_{image} is the image area size of a node. 1 is added to the denominator to avoid zero value.

For tag, we want a feature to capture how likely an HTML tag could indicate whether a node associated that tag is a ground truth node. This feature value is trained from the training data by a simple statistical calculation. we label this feature as TAG_SCORE.

We also consider tag density. The intuition is that main content and noisy content would be likely to have different tag density in a webpage. Suppose $numTags$ is the number of tags of a node (absolute feature value), and $numChars$ is the number of visible characters of a node (absolute feature value). We calculate the tag density as follows:

$$TAG\_DENSITY = \frac{numTags}{numChars + 1}$$

For link, we consider the link density. The intuition is that main content and noisy content would be likely to have different link density in a webpage. Suppose $numLinks$ is the number of links of a node (absolute feature value). We calculate the link density as follows:

$$LINK\_DENSITY = \frac{numLinks}{numTags + 1}$$

Now totally we have 17 features with adjusted values to learn a model. We will talk about how to learn a model in the next section.

5. LEARNING

To train the model, we need some labeled data. For a webpage, the ground truth regions could be marked by a set of DOM tree

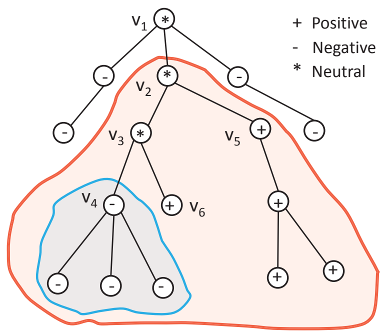

Figure 4: An example of DOM tree node labeling.

nodes. For some more sophisticated ground truth, inside the ground truth regions, there may be non-ground truth sub-regions, with each sub-region represented by a DOM tree node. As an example in Figure 4, node v_2 is marked by a user as a node to cover the region that he wants to select, and node v_4 is marked by the user as a node to represent the sub-region inside the selected region that he wants to cut off. The path from the root node to node v_2 is $[v_1 \rightarrow v_2]$, and the path from the root node to node v_4 is $[v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4]$. We define a positive node as the node which is part of ground truth for which all its descendant nodes are also part of the ground truth. We define a negative node as the node which is part of non ground truth and all its descendant nodes are also part of the non ground truth. The rest of the nodes in the DOM tree are the neutral nodes.

The following rules can be used to label the DOM tree nodes.

(a) A node which is deselected by the user must be labeled a negative node.

(b) All descendants of a negative node must also be labeled as negative nodes.

(c) A node is labeled as positive node if and only if one of its ancestors(including itself) is selected by the user and does not have a descendant which is deselected by the user.

(d) All descendant and ancestor nodes of a selected node (including itself) which are not labeled as negative or positive nodes by any of the rules in (a),(b) and (c) must be labeled as neutral nodes.

(e) The rest of the nodes which the rules in (a),(b), (c) and (d) do not apply to must be labeled as negative nodes.

Figure 4 is an example that shows the results after labeling the DOM tree nodes by applying the above rules. In this example, node v_2 is selected by the user and node v_4 is deselected by the user.

The "perfect" relative values of the position and area features for the ground truth regions, which we mentioned in Section 4.1, such as $BEST\_LEFT$, are set to be the average values of those relative feature values of all of the "top" ground truth nodes in the training data. A node is a "top" ground truth node if and only if it is a positive node and its parent is not a positive node. In Figure 4, nodes v_5 and v_6 are "top" ground truth nodes.

With a training dataset and the feature values, there are many classification models that could be trained to classify the nodes of a DOM tree. One of the strongest classification methods is SVM. However, even a very strong classification method would tend to misclassify many nodes, which we will demonstrate in Section 7.

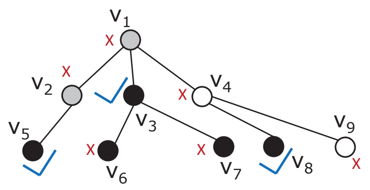

Figure 5: An example of candidate node compaction. The dark black nodes are the original candidate nodes with scores greater than the threshold.

So we want to choose a model with the ability of assigning good probability scores, rather than a binary classification. These scores will be used in the next steps for further selection and filtering before the final output. Hence the scores that can represent the probabilities are important. Although many methods can also generate scores along with classification, most of them are highly skewed, e.g., SVM. The Logistic Regression method can generate scores that are pretty much close to probability values. So we choose the Logistic Regression model in the learning step. We separate the training nodes into two classes for this model, with the positive nodes being one class and the rest nodes being the other class. After training the model, we can use the model to assign a score for each node. Assuming $x_1, \cdots, x_n$ are the feature values of a node v, after training a Logistic Regression model, its score can be set as:

$$p(v) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

The parameters $\beta_0, \beta_1, \cdots \beta_n$ are the model parameters trained from training dataset. The score $p(v)$ represents how likely node v is a ground truth (positive) node.

Candidate Node Selection

After obtaining the scores of all of the nodes in the DOM tree from the learning model, we select the initial candidate nodes with scores greater than a threshold. As we have mentioned earlier, a node can represent a region area covered by this node and all of its descendants. We can compact the candidate node set by removing those nodes with any of its ancestors being included in the original candidate node set. Figure 5 shows one example. v_3, v_5, v_6, v_7, v_8 are the original candidate nodes with scores greater than the threshold. Since v_6 and v_7 are descendants of v_3 which is also part of the candidate nodes, we remove them and only keep v_3, v_5 and v_8 as the candidate nodes.

6. GROUPING AND REFINING

6.1 Grouping

If we have a perfect score output and perfect score threshold, we can return the candidate nodes from the learning step and we are done here. However, web pages are full of noise and we would not be able to get perfect score output. There are very likely some noisy regions that are mistakenly picked and some missing regions. To further remove the noisy nodes and pick up the missing nodes, we apply a grouping technology. The heuristic is that users tend to select spatially consecutive regions, rather than picking regions here and there. This is also consistent with general web designs which usually put the main content and the advertisements and navigation bars in different spacial locations, rather than mixing them together. The candidate nodes from the output of a learning model

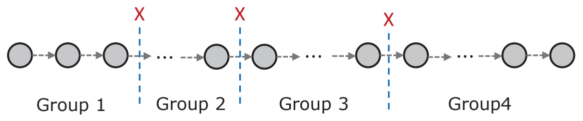

Figure 6: Separating the candidate nodes into different groups.

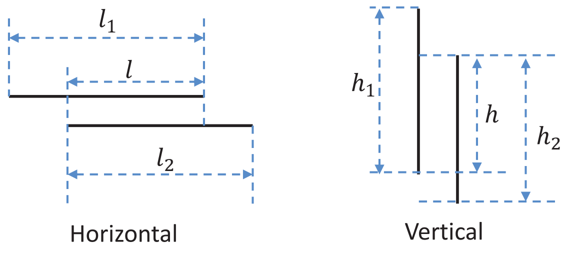

Figure 7: Horizontal and vertical position overlaps of two nodes.

may not all part of the main content, and some of them may belong to some other parts like advertisements. Our grouping idea is to use the physical spatial information and logical spatial information to try to separate the candidate nodes into different groups, with the target to put the nodes with close spatial connection into the same group, and then select a group with the highest possibility to be the main content.

First, we sort the candidate nodes by their positions in the depth first search of the DOM tree. We note that the depth first traverse of the DOM tree generally matches the same sequence of the nodes appearing in the webpage. In Figure 5, the candidate nodes after compaction are sorted in the ordered sequence of $[v_5 \rightarrow v_3 \rightarrow v_8]$. We can create links to connect consecutive nodes in the ordered sequence. To separate the nodes into different groups, we just need to break some of the links. In the end, those nodes which are still linked in the same sub-sequence are clustered into the same group. For this purpose, we just need to find the breaking points to break the ordered sequence to different sub-sequences. Figure 6 shows an example to use three breaking points to separate the candidate nodes into four groups.

To find the breaking points, we consider whether there is some overlap area for the neighborhood nodes in the ordered sequence, and how the horizontal and vertical projected positions of the neighborhood nodes overlap. Intuitively, if two neighborhood nodes have overlap area (which is very rare), they should be grouped together. Otherwise, we consider the projection overlap ratio. Figure 7 explains the projection overlap ratio. The horizontal lengths of the two nodes are l_1 and l_2, with projection overlap length l. The horizontal projection overlap ratio is set to be $min(\frac{l}{l_1}, \frac{l}{l_2})$. The vertical lengths of the two nodes are h_1 and h_2, with projection overlap length h. The vertical projection overlap ratio is set to be $min(\frac{h}{h_1}, \frac{h}{h_2})$. We define the projection overlap ratio (POR) to be the maximum value of the vertical projection overlap ratio and the horizontal projection overlap ratio. Intuitively, if POR of two nodes is high, they are aligned in positions and should be grouped together. Formally, for two nodes v_1 and v_2 in Figure 7,

$$POR_{1,2} = \max\left(min(\frac{l}{l_1}, \frac{l}{l_2}), min(\frac{h}{h_1}, \frac{h}{h_2})\right)$$

Suppose v_{i-1}, v_i, v_{i+1} and v_{i+2} are four consecutive nodes in the ordered sequence. With a predefined threshold T, we will break

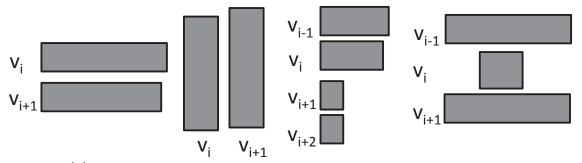

Figure 8: Projection overlap examples. v_i and v_{i+1} in both (a) and (b) have high projection overlap ratios; v_{i-1} and v_i, v_{i+1} and v_{i+2} have high projection overlap ratios in (c); v_{i-1} and v_{i+1} have high projection overlap ratios in (d).

the link between v_i and v_{i+1} if none of the four following conditions holds:

(1) v_i and v_{i+1} contain overlap area or $POR_{i,i+1} > T$.

(2) v_i and v_{i+1} have the same parent in the DOM tree.

(3) v_i and v_{i+2} contain overlap area or $POR_{i,i+2} > T$.

(4) v_{i-1} and v_{i+1} contain overlap area or $POR_{i-1,i+1} > T$.

Condition (2) allows two consecutive nodes with "logical alignment", i.e. they have the same parent, to be grouped together. Conditions (3) and (4) allow the node that is misaligned with its exact previous and its exact next neighbor nodes which are aligned to be grouped with its neighbors.

Figure 8 shows several examples for different projection overlap ratios. For (a) and (b) in Figure 8, since v_i and v_{i+1} have high projection overlap ratio, there will not be a breaking point between v_i and v_{i+1} in (a) and (b). For (c), since v_i and v_{i+1}, v_i and v_{i+2}, v_{i-1} and v_{i+1} do not have high projection overlap ratios, if v_i and v_{i+1} do not have the same parent in the DOM tree, then there will be a breaking point between v_i and v_{i+1} in (c). For (d), since v_{i-1} and v_{i+1} have high projection overlap ratio, there will not be a breaking point between v_i and v_{i+1} in (d)

One additional advantage for our grouping technology is that we only need to set one parameter, i.e. the projection overlap ratio, rather than considering a lot of parameters and features. Other useful features have been considered in the learning step. To make a good separation in the grouping step, the learning step and the candidate node compaction are necessary. After learning, a lot of noisy nodes have been discarded. And, after candidate node compaction, the redundant nodes have also been discarded. If the redundant nodes and too many noisy nodes are kept, it would be difficult to make a good separation in the grouping step.

6.2 Group Selection

After we separate the candidate nodes into different groups, we would need to further separate the groups into two category, one category for groups that belong to the main content, the other category for groups that belong to noisy content. As we notice that the main content is more likely to be spatially and logically a whole part, a good strategy is to select the best group. As the candidate nodes are generated from a learning approach, the majority area covered by the candidate nodes should belong to the main content. Otherwise it would indicate that the learning approach is not good and we need to improve the learning approach or switch to another learning approach. For the scores assigned by the learning approach, from the statistical point of view, the nodes that are more likely to be parts of contents should have higher scores. Hence we leverage both the area sizes and the scores to find the best group.

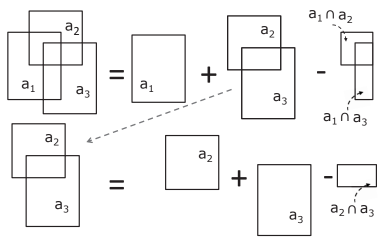

Figure 9: Example for computing area size.

For each group, we calculate the average score $\overline{P}$ of all its nodes from the Logistic Regression function output, and calculate the covered area size S. Then we pick up the best group by the largest value $\overline{P} \cdot S$.

To precisely calculate the covered area size S, first we collect all of the rectangles covered by the group. Suppose a_1, a_2, $\cdot$, a_n are the rectangles, $R_{a_1,a_2,\cdots,a_n}$ is the region covered by a_1, a_2, $\cdot$, a_n, and $f(R_{a_1,a_2,\cdot,a_n})$ is the area size of the region $R_{a_1,a_2,\cdot,a_n}$. Considering there may be some overlap between different rectangles, to calculate the covered area size by these rectangles, we can recursively apply the following equation:

$$\begin{aligned} & f(R_{a_1,a_2,\cdots,a_n}) \\ & \quad = f(R_{a_1}) + f(R_{a_2,a_3,\cdots,a_n}) - f(R_{a_1} \cap R_{a_2,a_3,\cdots,a_n}) \\ & \quad = f(R_{a_1}) + f(R_{a_2,a_3,\cdots,a_n}) - f(R_{a_1 \cap a_2, a_1 \cap a_3, \cdots, a_1 \cap a_n}) \end{aligned}$$

As $a_1 \cap a_2$, $a_1 \cap a_3$, $\cdots$, and $a_1 \cap a_n$ are also rectangles, both $f(R_{a_2,a_3,\cdots,a_n})$ and $f(R_{a_1 \cap a_2, a_1 \cap a_3, \cdots, a_1 \cap a_n})$ can be further recursively decomposed using the above equation. Figure 9 shows an example to calculate the area size for three overlapped rectangles. The area size of region R_{a_1,a_2,a_3} is equal to the area size of region R_{a_1} plus the area size of region R_{a_2,a_3} minus the area size of region $R_{a_1 \cap a_2, a_1 \cap a_3}$. And the area size of region R_{a_2,a_3} is equal to the area size of region R_{a_2} plus the area size of region R_{a_3} minus the area size of region $R_{a_2 \cap a_3}$. Similarly, the area size of region $R_{a_1 \cap a_2, a_1 \cap a_3}$ is equal to the area size of region $R_{a_1 \cap a_2}$ plus the area size of region $R_{a_1 \cap a_3}$ minus the area size of region $R_{a_1 \cap a_2 \cap a_1 \cap a_3}$.

6.3 Refining

After the "Best Group" is selected, most of the noisy nodes are expected to be removed. However, the selected group may not always be able to contain all of the nodes of the main content. One part of the reasons is that It is impractical to train a perfect learning model for this problem. The better the output from the learning model, the less likely it would be to miss the nodes of the main content in the selected group. Another part of the reasons comes from diversity of the web page designs. The grouping technique would not be able to perfectly work for all kinds of web pages. However, based on the criterion of group selection, the nodes in the"Best Group" still have the highest possibility to be the main content among all of the groups. As we have already mentioned, the main content tends to be logically and spatially a whole part. So, to pick the missing nodes of the main content, we can expand from the "Best Group".

Our strategy is to expand the group by replacing the nodes with their common lowest ancestor if the area size of the "Best Group" is too small. And, we recursively replace the common lowest ancestor with its parent if the area size covered by the node is still too small and it is not very close to the root node. If area size of the common lowest ancestor or a higher level of ancestor that is selected is too big, it might be likely to introduce new noisy content. We then recursively drill down to replace with the child node with the biggest area size among all of children so long as the area size is not too big. Then, from the the siblings of the child node and the child node itself, we pick up the best node by considering both score and area size. Algorithm 1 describes the details of the refining process. We select the parameters in the algorithm based on the statistical analysis of the training data. We will explain with more details in Section 7.

After the refining steps, we also do some minor checking to see if the title is obviously missing from the selection. If it is missing, we add the title node into the final selection. There are some literature work for title detection [6]. We only check *H1* to *H6* tags, where *H1* is most preferred and *H6* is least preferred. If a most preferred title candidate node exists in the location of the top of the main content selection, we add it to the final selection.

Algorithm 1 Refining the candidate nodes

```
Input: A list of candidate nodes, DOM tree
Parameters: area size ratios $\theta_1$, $\theta_2$, $\theta_3$, $\theta_4$; level depth $d$
Output: A list of DOM tree nodes
1. Get the common lowest ancestor $v$ of the candidate nodes in the
   DOM tree. Calculate the overall covered area size $S_1$ of node $v$,
   i.e. $S_1 = s(v)$
2. If $S_1 < \theta_1 \cdot S_{page}$, where $S_{page}$ is the area size of the whole web
   page, repeat: $v \leftarrow parent(v)$, $S_1 = s(v)$, until $S_1 \geq \theta_2 \cdot S_{page}$.
3. Get the refined nodes differently according to different conditions:
   (a) If the depth of node $v$ in the DOM tree greater than $d$, or
       $S_1 \leq \theta_3 \cdot S_{page}$, return node $v$ as the output and stop here.
   (b) Else if $S_1 \geq \theta_4 \cdot S_{page}$:
       (i) Repeat: $v \leftarrow \arg\max_{v'} \{s(v') | v' \in children(v)\}$,
           $S_2 = f(v)$, until $S_2 < \theta_4 \cdot S_{page}$
       (ii) $v_{out} \leftarrow \arg\max_{v'} \{p(v') \cdot s(v')\}$ , where $v' \in$
            $siblings(v) \cup \{v\}$
       (iii) return $v_{out}$ as the output
   (c) Else, return the original candidate nodes.
```

7. EXPERIMENTAL EVALUATION

7.1 Evaluation Data Set and Metrics

Evaluation Data Set

We use the log data from the real product HP SmartPrint for evaluation. The log data records the clip activities of those users who agree to provide their clip data to SmartPrint for the purpose of improving the user experience of the product. In the log data, the information includes clip paths, the urls of the web pages etc. We downloaded and parsed the webpages using the webkit rendering engine. We chose the clip data that have been manually selected by users. As the web pages may have been changed since the clip data is recorded, we excluded the data with any of clip paths not matching any of the paths in the web page. With this preliminary step of filtering, we filtered some noisy data. Then we asked people

Table 1: Statistics of the data set

Total number of pages	2000
Number of training pages	1335
Number of testing pages	665
Number of websites (domains)	805

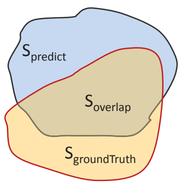

Figure 10: The region overlap.

to examine the ground truth and filter out the obviously non-ground truth data which might be caused by non-intentional user behaviors. At last, we have a clean dataset for training and evaluation. The total number of pages is 2000, which comes from 805 websites, i.e., there are 805 domain names in this dataset, making it to be sufficiently diversified rather than being biased on a small number of websites. We used two thirds of the dataset with 1335 pages as the training dataset and one third of the dataset with 665 pages as the testing dataset. The statistics of the dataset are shown in table 1.

Evaluation Metrics and Parameters

The recommended content regions are the output from the algorithm. The effectiveness of the recommendation can be measured by how close the recommended regions and the ground truth regions are matched. We use the precision and recall and F-measure as the evaluation metrics. To show how to calculate the metrics, we create an example as shown Figure 10. The light blue area refers to the region selected by a user and the light red area refers to the region recommended by a method. Suppose the overlap area size is $S_{overlap}$, and the area size selected by the user is $S_{groundTruth}$, and the area size recommended by the method is $S_{predict}$. Then we can calculate the precision and recall and F-measure as the follows:

$$P = \frac{S_{overlap}}{S_{predict}},\ R = \frac{S_{overlap}}{S_{groundTruth}},\ F_1 = \frac{2PR}{P+R}$$

The parameters we choose are as follows:

Logistic Regression Score Threshold = 0.55, Projection Overlap Ratio Threshold = 0.70. We will report on a sensitivity analysis for these two parameters in the next subsection.

$\theta_1 = 0.05$, $\theta_2 = 0.2$, $\theta_3 = 0.4$, $\theta_4 = 0.9$, $d = 3$ in Algorithm 1. Based on the training dataset, we found that around 1% of the pages have the ground truth area size which is less than 5% of the page area size; less than 2% of the pages have the ground truth area size greater than 90% of the page size; and the ground truth area size distribution has the highest distribution density in the range from 20% to 40% of the page area size. That's the reason why we pick these parameters for the refining step in Section 6.3. d value is based on the general web page designs, where we observe that the levels of the main content nodes are usullay not the top 3 levels.

7.2 Comparison with the Baseline Methods

We consider the following baseline methods:

- **LR_A:** This baseline method applies the logistic regression learning model to compute the scores for all of the nodes of the DOM tree for a webpage, and only selects the node with the highest score as the output node.
- **SVM_A:** This baseline method applies the linear SVM learning method to learn a linear model and utilizes the model to compute the scores for all of the nodes of the DOM tree for a webpage, and only selects the node with the highest score as the output node.
- **LR:** This baseline method applies the logistic regression learning model to compute the scores for all of the nodes of the DOM tree for a webpage, and only selects the nodes with the scores higher than the threshold and removes the redundant nodes. As mentioned previously, a redundant node is a node in which at least one ancestor node is included in the candidate node set.
- **SVM:** This baseline method applies the SVM learning method to classify the nodes of the DOM tree for a webpage and selects the nodes that are classified as positive with redundant nodes removed. We select the linear kernel for SVM since we have totally 964380 training nodes which is huge and it is computationally expensive to train an SVM model on a large training items with a nonlinear learning kernel.
- **MSS:** This baseline method applies the Maximum Subsequence Segmentation algorithm proposed in [13]. We use the implementation of this method in [4] for comparison.

We label our proposed method, which utilizes the combination of learning and grouping, as **CLG**. The use the same features as described in Section 4 for all methods except *MSS*. For method *MSS*, we use the set of features described in literature [4]. We use the identical score threshold parameter for *LR* and *CLG*. We will further analyze the parameters in the next subsection.

Figure 11 reports on the precision and recall values for all of the methods, i.e., *LR_A*, *SVM_A*, *MSS*, *LR*, *SVM* and our proposed method *CLG*. As expected, since *LR_A* and *SVM_A* pick only one node that has the highest score from the learning models, they have higher precision values than all other methods. However, their recall values and F_1 values are very low, this is because the area of the node with the highest score is very likely to be just a small fraction of the whole ground truth area. So, *LR_A* and *SVM_A* are not the good solutions. Method *MSS* show pretty much lower recall value and F_1 value than *LR*, *SVM*, and *CLG*, although its precision value is kind of close to the precision values of other methods. As shown in literature [13, 4], this method works very well for article webpages. However, our experiment shows that it is not that good for general webpages. This tells us that a method that works well for article web pages might not work well for general web pages. Method *CLG* has higher F_1 value than all of the baseline methods. Method *CLG* also has higher precision, recall and F_1 values than methods *LR* and *SVM* both of which utilize learning step only. The learning model for *CLG* and *LR* are identical. This demonstrates that the grouping step is valuable and helps to significantly further improve the results from the learning step.

Figure 12 is an example that shows the selected regions by different methods for an identical recipe web page. For this web page, users would be interested in the main content in the middle region. Methods *LR* and *SVM* are both able to pick the majority of the main content, but at the same time, they also pick pretty many noisy content regions which are scattered around the page. If a tool makes human effort capable to further clean the selection, that would be pretty much work to remove the scattered noisy content regions.

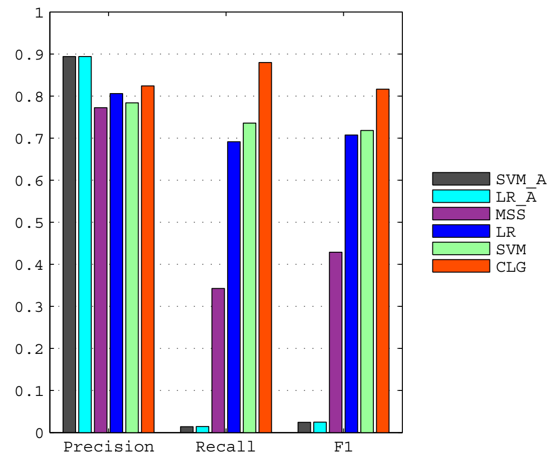

Figure 11: Precision, recall and F_1 measure for different methods.

MSS method is able to pick up some regions that are almost logically continuous, from part of the middle block, to the right and then to to the bottom blocks. Continuous regions are good for users to do further manual selection. However, *MSS* method misses more actual content and mistakenly picks more noisy content. Method *CLG* selects exactly the main content for this web page, which is continuous and complete. This example further demonstrates that our solution has higher quality of content selection than the baseline methods.

As *CLG* tends to select the regions which are spatially a single part, even if the output is not exactly what a user wants, it will be easy for the user to make further manual selection, which is convenient for clipping webpages for e-reading or other purposes.

7.3 Parameter Sensitivity Analysis

We want to analyze how the parameters will impact the results.

Sensitivity Analysis for Score Threshold

Figure 13 shows the curve of the precision and recall values with respect to different score thresholds to pick up the original candidate nodes for the method *CLG*. As expected, when the threshold value increases, the precision value increases and the recall value decreases. However, the curve does not show steep shape. Especially when the score threshold is greater than 0.4, the precision and recall values are pretty stable, which means that the precision and recall values are not sensitive to the score threshold value. We select the threshold value to be 0.55 for a good balance. Figure 14 compares the F_1 values for the methods *LR* and *CLG* with respect to different score thresholds for the learning step. The figure shows that *CLG* has higher F_1 values for all score thresholds, and the F_1 values of *CLG* are much more stable than the F_1 values of *LR*. The F_1 value of *LR* shows significant decrease when the score threshold value is greater than 0.6. This demonstrates that the grouping step of *CLG* helps to make the selection output more accurate and less sensitive to the score threshold parameter.

Sensitivity Analysis for Projection Overlap Ratio Threshold

For the projection overlap ratio threshold described in Section 6.1, we would expect that when the threshold value increases, noisy nodes and actual content nodes would be less likely to be grouped into the same group, which would help to increase the precision. However, with the increase of the projection overlap ratio threshold, actual content nodes would be more likely to be grouped into

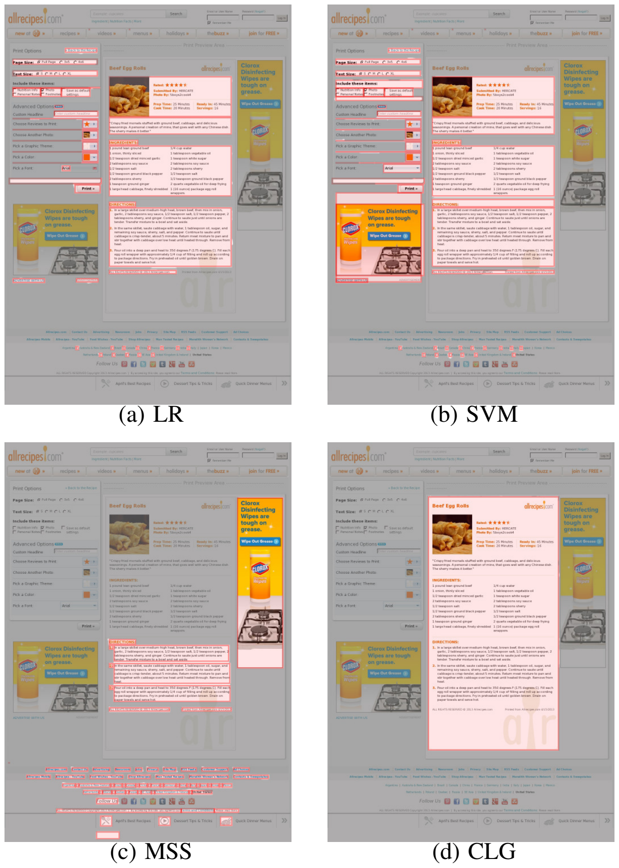

Figure 12: An example of selection output by *LR*, *SVM*, *MSS*, and *CLG*. The regions with light red background are the selected regions.

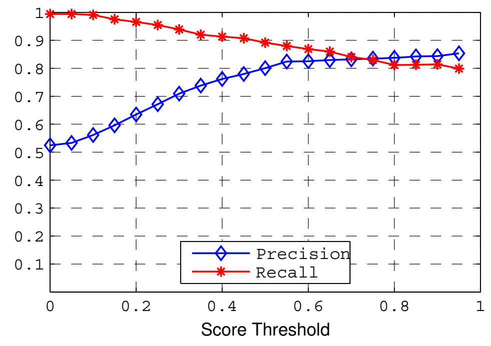

Figure 13: Precision and recall values for *CLG* with respect to different logistic regression thresholds.

different groups. We should pick a good threshold value to make a good balance. Figure 15 reports on the precision and recall values for *CLG* with respect to different projection overlap ratio thresholds. For a good balance, we have picked a threshold value of 0.7 for other experiments. As expected, When the threshold increases, the precision value increases and the recall value decreases. We also notice that both values do not change significantly. This demonstrates that the output of *CLG* is not sensitive to the projection overlap ratio threshold.

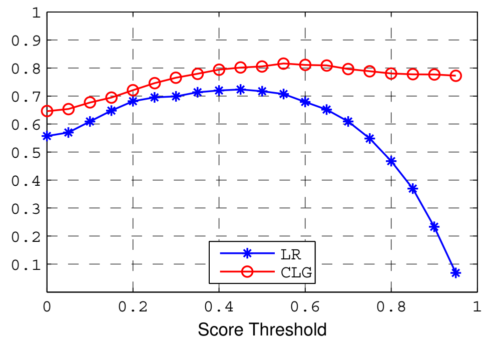

Figure 14: F_1 values for *LR* and *CLG* with respect to different logistic regression thresholds.

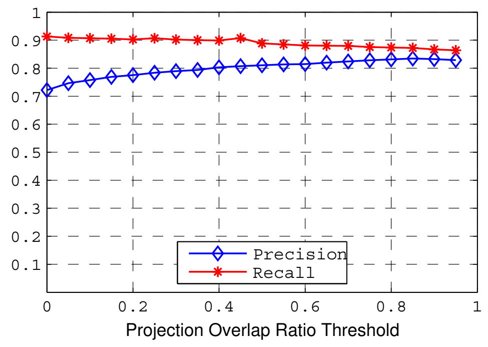

Figure 15: Precision and recall for *CLG* with respect to different projection overlap ratio thresholds.

8. CONCLUSIONS

In this paper, we propose an effective approach by combination of a learning model and a grouping technology to identify actual content from web pages. We generate multiple features by utilizing DOM tree node properties to train a machine learning model and select candidate nodes based on the learning model. With the observation that the actual content is usually located in a spatially continuous block, we develop a grouping technology to further process the candidate nodes by filtering out noisy data and picking missing data. We show the effectiveness of our solution in a real dataset.

9. REFERENCES

[1] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 580–591, 2002.

[2] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. Extracting content structure for web pages based on visual representation. In *Proceedings of the 5th Asia-Pacific web conference on Web technologies and applications*, APWeb'03, pages 406–417, Berlin, Heidelberg, 2003.

[3] L. Chen, S. Ye, and X. Li. Template detection for large scale search engines. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 1094–1098, 2006.

[4] J. Fan, P. Luo, S. H. Lim, S. Liu, P. Joshi, and J. Liu. Article clipper: a system for web article extraction. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 743–746, 2011.

[5] D. Fernandes, E. S. de Moura, B. Ribeiro-Neto, A. S. da Silva, and M. A. Gonçalves. Computing block importance for searching on web sites. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 165–174, 2007.

[6] Y. Hu, G. Xin, R. Song, G. Hu, S. Shi, Y. Cao, and H. Li. Title extraction from bodies of html documents and its application to web page retrieval. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 250–257, 2005.

[7] H.-Y. Kao, S.-H. Lin, J.-M. Ho, and M.-S. Chen. Mining web informative structures and contents based on entropy analysis. *IEEE Trans. on Knowl. and Data Eng.*, 16(1):41–55, Jan. 2004.

[8] C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 441–450, 2010.

[9] C. Kohlschütter and W. Nejdl. A densitometric approach to web page segmentation. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 1173–1182, 2008.

[10] S. H. Lim, L. Zheng, J. Jin, H. Hou, J. Fan, and J. Liu. Automatic selection of print-worthy content for enhanced web page printing experience. In *Proceedings of the 10th ACM symposium on Document engineering*, DocEng '10, pages 165–168, 2010.

[11] S.-H. Lin and J.-M. Ho. Discovering informative content blocks from web documents. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 588–593, 2002.

[12] P. Luo, J. Fan, S. Liu, F. Lin, Y. Xiong, and J. Liu. Web article extraction for web printing: a dom+visual based approach. In *Proceedings of the 9th ACM symposium on Document engineering*, DocEng '09, pages 66–69, 2009.

[13] J. Pasternack and D. Roth. Extracting article text from the web with maximum subsequence segmentation. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 971–980, 2009.

[14] D. C. Reis, P. B. Golgher, A. S. Silva, and A. F. Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 502–511, 2004.

[15] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning block importance models for web pages. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 203–211, 2004.

[16] A. Spengler and P. Gallinari. Document structure meets page layout: loopy random fields for web news content extraction. In *Proceedings of the 10th ACM symposium on Document engineering*, DocEng '10, pages 151–160, 2010.

[17] F. Sun, D. Song, and L. Liao. Dom based content extraction via text density. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 245–254, 2011.

[18] J. Wang, C. Chen, C. Wang, J. Pei, J. Bu, 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 ACM SIGKDD international*

conference on Knowledge discovery and data mining, KDD '09, pages 1345–1354, 2009.

[19] T. Weninger, W. H. Hsu, and J. Han. Cetr: content extraction via tag ratios. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 971–980, 2010.

[20] L. Yi, B. Liu, and X. Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 296–305, 2003.

[21] L. Zhang, L. Tang, P. Luo, E. Chen, L. Jiao, M. Wang, and G. Liu. Harnessing the wisdom of the crowds for accurate web page clipping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 570–578, 2012.