Automatic User Comment Detection in Flat Internet Fora

Mathias Bank
Faculty for Mathematics and Economics
University of Ulm
Ulm, Germany
mathias.bank@uni-ulm.de

Michael Mattes
Faculty for Mathematics and Economics
University of Ulm
Ulm, Germany
michael.mattes@uni-ulm.de

***Abstract*—Millions of people are using the World Wide Web and are publishing content online. This user generated content contains many information relevant not only to marketing but to companies in general (customer-oriented products), governments (direct democracy) and many more. Analysis on such data becomes more and more important. This paper deals with a prerequisite: we propose an algorithm to automatically detect posting structures in flat internet fora to extract user comments. The algorithm is able to handle a wide range of different fora systems — even nested structures. The approach first detects the main content section by applying a modified version of the SST algorithm and then detects the posting structure by using several posting properties found in internet fora. It creates XPath expressions for faster data extraction in further steps.**

***Keywords*-social media, internet community, forum, web 2.0, Information Retrieval, crawler, extraction**

I. Introduction

User generated content becomes more and more important. There are 475 million active internet users in the world. More than 38% want to start their own weblog [1]. In Germany, there are 42.7 million internet users [2] and 35% of them are interested in publishing their own content [3]. 78% of the internet users rely on recommendations from consumers and 61% trust customer opinions posted online [4]. That is why user generated content becomes more and more important. The huge number of user postings makes it necessary to analyze them automatically. The prerequisite is to be able to extract user comments from online data.

There is a wide range of different community systems. Besides classical systems like newsgroups and internet fora there are so called Web 2.0 systems like weblogs, twitter, flickr or youtube. In this paper, we will focus on flat internet fora. The goal is to extract user postings directly from discussion pages. This could simply be done by downloading and storing complete discussion pages, but the approach would lead to noisy data because we would have to handle non-relevant content. An enhanced extraction algorithm should only store user comments and meta information themselves without navigation or advertisement and makes information extraction and data mining tasks more efficient.

The World Wide Web contains thousands of different internet fora. We present an unsupervised approach which is able to handle many of them. The proposed algorithm is able to detect posting structures — including the user comment and meta information (user name, timestamp) — automatically. This is done in two steps: first the interesting content section is identified, then this section is divided into different postings. The algorithm creates an XPath[1] wrapper to further increase the extraction performance.

We assume to have valid HTML or XHTML pages represented as DOM trees. Most of the currently available community systems do not create valid pages, therefore we have to use cleaning algorithms like the ”tidy” algorithm to make an analysis possible.

II. Related Work

Unsupervised content extraction algorithms can mainly be categorized into local and global techniques [5]. Local algorithms just use a single page and try to detect the relevant content involving implicit or explicit assumptions about how the content is represented. Global algorithms instead use many pages to detect the relevant content by comparing these pages with each other. There have been many researchers dealing with the problem of content detection using local and global algorithms. [6] gives a good overview about some of them.

In almost all community systems pages are generated with the help of template systems. This fact leads to the assumption that template structures can be detected by comparing discussion pages. There is a wide range of possible algorithms: we have chosen the Site Style Tree (SST) approach [7] as global extraction technique.

The SST algorithm was designed to detect reccuring sections on web pages. The base concept, the Site Style Tree, is made up of two different node types: style nodes and element nodes (fig. 1). An element node represents a DOM node with tag, attributes and child information, a style node is a list of element nodes for style representation. For each web page — represented by its DOM tree — the algorithm creates a Style Tree by recursively creating element nodes for DOM nodes and grouping their children to style nodes. Merging these Style Trees and storing the information, how often a style node can be found in an element node, creates the Site Style

[1]W3C standard for addressing parts of XML documents. Version 1: http://www.w3.org/TR/xpath/

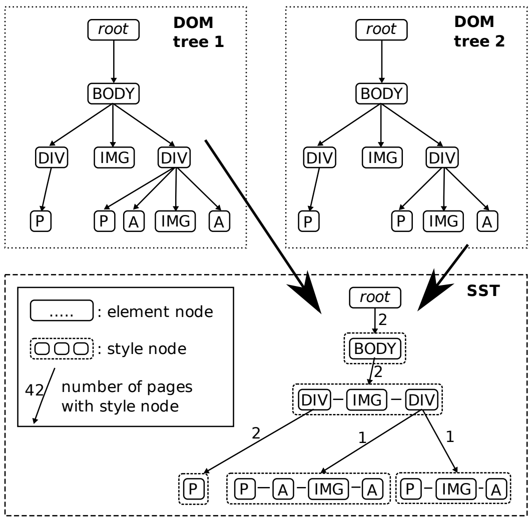


Fig. 1. The SST algorithm transforms a list of DOM trees into a Site Style Tree consisting of style and element nodes. These elements are used to identify redundant page structures. Using Shannon's entropy the algorithm computes values indicating the importance for every DOM node.

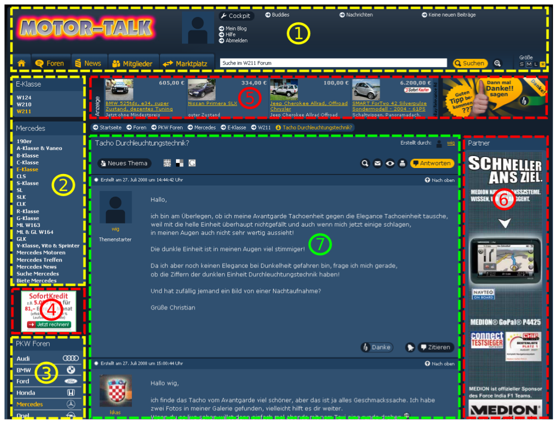


Fig. 2. A discussion page consists of several sections: the header (1), the category list (2, 3), advertisement (4, 5, 6) and the content section (7) (sample taken from http://www.motor-talk.de).

Tree. This tree is used to compute node relevances with two different relevance indicators: the content relevance considers textual differences between element nodes, the presentation relevance assesses contained style nodes: the more style nodes an element node has, the more important it is. Using Shannon's entropy [8], the algorithm combines these two relevance measures and assigns an importance value to each node $\in [0, 1]$. Nodes with identical content and substructure over several pages receive low importance values. Nodes with different content and substructure get higher ones. We will use these values in our approach. Due to page limitations, we have to point to the original paper [7] for more details.

Extracting user postings from internet fora is quite more than just detecting the relevant content. An important task is to detect each user posting. This step of content segmentation is already focused by many researchers. According to [5]'s technique categorization, they deal with local algorithms which make assumptions about repeating structures. There have been two observations on record lists [9]:

1) Data regions are presented in continuous regions and have a similar HTML tag structure (*data region*).
2) Web pages are represented with HTML tags, which form a tag tree. A data region can be found in one subtree.

[9], [10] and [11] try to automatically extract product information from web pages by dividing a DOM structure into similar substructures. The similarity of substructures can be calculated in different ways. The simplest way is to allow only identical substructures. [11] introduced a Tree Edit Distance to compensate small differences in substructures. [12] extended this method to be more flexible by weighting node types differently. All these methods are constructed to detect lists. [13] focused on detecting nested structures by post-order traversing a tag tree which we can think of as a special DOM tree. [14] improved this method by using more visual information like gaps and spaces.

III. User Posting Detection

Detecting user generated content in community systems is a complex task that can be divided into two subtasks:

1) main content detection
2) posting segmentation

We will discuss these subtasks separately. Each one will create an XPath wrapper to increase data extraction performance in further tasks.

A. Main Content Detection

Community systems have many different sections on discussion pages. Typically, there is a header section, a navigation section, advertisement sections and the content section (see fig. 2). We use a modified version of the SST algorithm to detect the content section from which we want to extract user postings.

There are two different page types in internet fora: discussion overviews and discussion pages. If we apply the SST algorithm to all pages at once, we cannot expect to get good results because of different template designs. Therefore, the SST algorithm must be applied to each page type separately. Using the generated importance values on discussion pages, we create a wrapper to get an XPath expression which extracts the content section:

- The header and advertisement sections exist on every page with nearly the same content. They get a low importance value.
- The content section instead changes more frequently — there is no second page with the same content. This section gets the highest importance value.

We traverse the complete Style Tree for every discussion page and collect the nodes with the highest importance. Each one is used to create an XPath expression which is saved in a list. There are three types of wrong expressions:

1) *Trivial expressions:* In our tests, there have been some pages where the body tag was marked as most important due to importance propagation from its children. We assume that this node cannot be the correct content node and that there has to be at least one navigation or header section. These expressions are discarded by the algorithm.
2) *Too specific expressions:* Discussion pages consist of different numbers of postings. Especially if there is just one posting, the generated XPath expression is more specific than the ones generated on pages with more postings. This expression is not wrong at all, it is too specific. So we have to generalize it by comparing the XPath expression with expressions generated on other pages. If there is an expressions with lower precision[2], we have to use this one.
3) *Wrong expressions:* In some cases, the most important node is not a content node. This appears, if there is a section that does not exist on other pages or if the section has a completely different content. These expressions can be detected using a simple majority vote.

The algorithm can eliminate all wrong XPath expressions automatically so that only one correct XPath expression remains which points directly to the content section.

B. Post Segmentation

After finding a direct path to the content section, we divide it to detect the complete posting structure composed of the user comment and meta information (e.g. user name, timestamp). Other scientists focused on the segmentation of product pages. The task of comment segmentation has not been considered until now.

We have analyzed more than $3,500$ real postings in 13 different internet fora based on 8 fora systems and have done some interesting observations in addition to those mentioned in [9]:

1) User comments mainly consist of textual data which spans more than one line. At an average, a post has 320 characters (Median: 187, 0.25 quantile: 80.3, 0.75 quantile: 387) with 4.3 line breaks (Median: 2, 0.25 quantile: 1, 0.75 quantile: 6). Postings without line breaks (25%) have an average of 94 characters (Median: 51, 0.25 quantile: 17, 0.75 quantile: 104.3).
2) Paragraphs are represented either with `p` nodes or they are separated by `br` nodes.
3) The complete user comment can be found in one parent node which is possibly splitted by further elements: `a`, `img`, `code`, `object`, `blockquote` and sometimes `ul` or `ol` nodes.
4) Most community systems format quote and code sections with `div` or `table` substructures. These substructures may also split the user comment.
5) User comments are different. Comparing complete postings (with user comment, user name and timestamp) we can see that there is no similarity except the base posting structure, which encloses the user comments. We are looking exactly for this base structure.
6) Some community systems use nested structures to represent answer relations requiring us to use an algorithm that is able to detect nested structures.

[2]The precise XPath expression contains the more general one.

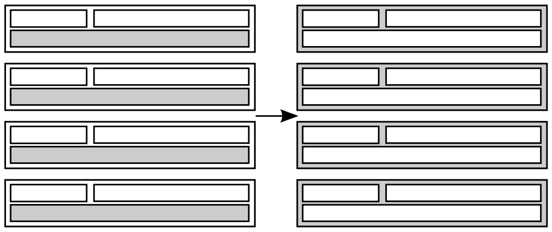


Fig. 3. Post candidate search: After detecting user comments (left side) with the help of text nodes a node generalization returns the complete posting structure (right side).

The algorithms found in literature do not use these properties. They are not capable of posting detection because the algorithms have to deal with different substructures in depth and length. Our algorithm for post segmentation works differently and takes advantage of these observations. It is a mixture of local and global detection techniques and uses the base idea of [13]: traversing the content section in a bottom up way to detect nested structures.

1) Post Candidate Search: The proposed approach first looks for possible posting candidates which mainly consist of user comments themselves. We observed that these comments can be found in one parent node. The task of post candidate search is to find these comments and to generalize them to the complete posting structures (fig. 3). This is done in four steps:

Merge inline elements: User comments are separated by several tags. Most of them are inline elements[3] — so called ”text level” elements. We extend these elements with block elements that are typically used in comment sections but not for posting structures: `pre`, `blockquote` and `p`. The algorithm moves their content to the corresponding parents and removes the nodes themselves. We get a modified DOM tree with block elements only.

Find text nodes: In the modified DOM tree, the algorithm looks for possible comment candidates. It takes advantage of the observations made and uses the following criteria that have shown to return good results:

- The text node consists of at least 150 characters or
- the text node consists of at least 50 letters and new line characters (`\r`,`\n`).
- The text node does not begin with ”____”, which is typically used for signatures.

The parents of these text nodes are candidates c_i for user postings and are collected in a list L. It is important to notice

[3]We use the W3C definition of inline elements: http://www.w3.org/TR/REC-html40/struct/global.html#h-7.5.3, namely `span`, `img`, `b`, `i`, `strong`, `em`, `a`, `acronym`, `abbr`, `code` and `br` nodes

that this list is not necessarily complete because the algorithm misses very short entries. They will be recovered afterwords using the generated XPath expression. The list L can also contain unwanted nodes like quote or code sections that have not been detected as comment elements (because `div` or `table` substructures have been used). The criteria just ensure that posting structure elements or meta data (e.g. user name or timestamp) are not selected. But they can cause two different possible mistakes: if the user comment itself is too small to be selected and the contained quote / code block is big enough, we get a completely wrong candidate. If the user comment instead also reaches the necessary text length, we get more candidates than real user postings. The next two steps will handle these problems.

Clean text nodes: Not all candidates $\in L$ are user comments. In some circumstances, a text node is a subelement of a bigger comment (e.g. quote or code block). We have to remove wrong candidates by using the property that the complete comment can be found in one parent node. For each candidate c_i, we recursively select all parents $p_1 \dots p_n$. If we find a parent p_i in the candidate list L, we remove the candidate c_i since the list contains a more general text node c_j already.

Node generalization: All direct text node parents are stored in L. They do not represent complete posting structures yet because additional meta information like time and user name is missing. The algorithm has to generalize comment nodes $\in L$ to find the complete posting structure.

The observations have shown that each posting can be found in its own subtree. The cleaning step has already ensured that there is only one candidate for each posting (or none). To find the posting structure, the algorithm just has to select more general nodes:

```
listChanged = true;
while listChanged do
    listChanged=false;
    for candidate $c$ in $L$ do
        if $c$ has a parent $p$ and this is the only one then
            replace $c$ with $p$ in $L$;
            listChanged=true;
        end if
    end for
end while
```

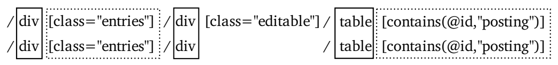
The generalization is done for all candidates by replacing each candidate c_i with its parent node p_i until one of the following stopping criteria is true:

1) p_i does not exist
2) p_i is also parent of another candidate $c_j \in L, j \neq i$

With these stopping criteria we ensure to get no overlapping while generalizing candidates. The problem of too specific text node candidates due to quote or code sections is resolved automatically by this step.

2) Wrapper Generation: The candidate list L can now be used to generate an XPath wrapper to automatically select posting nodes in further extraction steps.

We use a global detection strategy to calculate the correct XPath expression for posting structures. For each candidate $c_i \in L$, we create the corresponding XPath expression as detailed as possible by using all available attributes without position information (e.g. `/div[@id="content"][@style="color:black"]`). Collecting all XPath expressions allows to compare subpatterns (fig. 4): we divide each XPath expression into subpatterns by splitting the XPath string at slash characters "/". If all expressions have the same node type at position i, we add this pattern with the common node attributes to the resulting XPath. The XPath creation is stopped if one node type is different or if all subpatterns have been checked.



```
/div[class="entries"]/div[class="editable"]/table[contains(@id,"posting")]
/div[class="entries"]/div                  /table[contains(@id,"posting")]
```

Fig. 4. Wrapper generation: All subpatterns in all XPath expressions are compared to each other. The resulting XPath expression contains equal node types and attributes.

TABLE I
XPATH EXPRESSION QUALITY

	content section	posting segmentation
perfect	82.4%	64.0%
correct	100%	90.0%

In nested structures, the process is altered by inverting the subpattern list to check nodes backwards. Identical nodes are added at the front of the generated XPath expression and the algorithm stops by adding an additional slash character "/" at the front to make different "root" nodes possible.

This global strategy ensures to generate an XPath expression for all postings. Using this expression, we will get all postings, even the ones missed in previous steps.

IV. EVALUATION

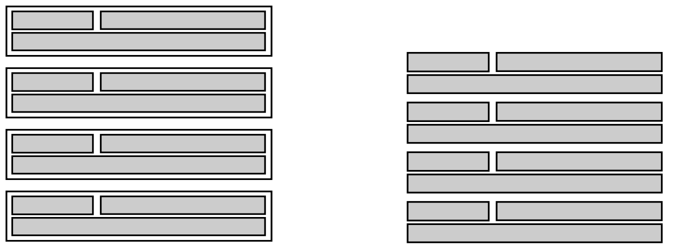
The complete algorithm creates two different XPath expressions. The first one ensures, that we only detect posting structures in the main content section. The second one points to posting structures in this main content section. In our evaluation, we check both expressions.

The algorithm was tested against 51 real internet fora, based on 14 different fora systems. Each forum uses a different template and was not used except for evaluation. So we can also check our observations. The evaluation was generated by loading 20 discussion pages per forum and applying the proposed algorithm. The resulting XPath expressions have been checked for their suitability (table I). An XPath expression for the content section is perfect, if it points directly to the parent nodes of postings. It is correct, if the content section is within the selected subtree. An XPath expression for posting segmentation is perfect if it only returns postings. It is correct if it points to postings but also to other elements.

Content detection works great. Posting segmentation works good but could be further improved. It is able to compensate the preciseness of the content detection. Comparing the results depending on forum systems (table II) shows that systems with

TABLE II
SYSTEM BASED QUALITY ANALYSIS

system	quantity	correct	perfect
Burning Board	3	100%	66.7%
drupal	6	83.3%	0%
IPB	5	75%	50%
myBB	4	66.7%	66.7%
phpBB	7	100%	57.1%
SMF	3	100%	33.3%
Unclassified NewsBoard	2	100%	100%
Vanilla	3	100%	100%
vBulletin	13	92.3%	84.6%
miscellaneous	5	60%	60%


(a) Postings with own subtree

(b) Postings without own subtree

Fig. 5. The proposed algorithm is able to detect posting structures if the complete posting (user comment and meta information) can be found in an own subtree (a). Some templates do not follow this assumption and present meta information and user comments of all postings in one subtree (b).

additional content in the content section are difficult to handle. Discussion specific sections with topic related information can cause problems: the generated XPath expression for posting structures could detect non-postings. Before creating the XPath wrapper, the algorithm is aware, which sections are postings and which are not. It is very hard to store this information in XPath expressions automatically (with the help of siblings or children). In our application, we want to extract user name, posting time and user comments to apply text mining methods. To get the correct sections, we reapply the SST algorithm to extracted postings themselves. Only changing sections such as the user comment, the user name and the posting time get a high relevance. Discussion headers, footers and advertisement are discarded easily. This is why correctness suffices for our approach in most cases.

The algorithm is not able to detect user postings if they cannot be found in own subtrees (fig. 5) because the assumption of [9] is not valid. The abstraction step fails. It is possible to detect such structures by testing different node combinations and creating virtual nodes. This was not done in this work.

Our approach is also not able to handle systems in which the first comment is formated completely different (similar to weblog systems). The global strategy in our wrapper generation would have to check XPath expressions at same posting positions to solve this problem. Until now, we have not found a really good solution to allow special representations at any position. Fortunately, we only have seen this style in few individual and drupal based systems.

Last but not least, the algorithm is not applicable to community systems with low answer rate, because we need more than one posting to make the generalization step possible.

V. CONCLUSION

The proposed algorithm has shown that it is possible to detect posting structures automatically in flat internet fora. It is applicable to a large number of different systems and templates if the posting can be found in an own subtree.

Unfortunately, the algorithm cannot be applied completely unsupervised because the user has to check the applicability. Further research on the topic of user comment extraction should address one important question: Is it possible for an extraction algorithm to pre-check a web page for a suitable structure or can the algorithm post-check the reported postings for plausibility?

REFERENCES

[1] U. McCann, "Wave.3 - social media tracker," March 08. [Online]. Available: http://www.universalmccann.com/

[2] B. van Eimeren and B. Frees, "Ergebnisse der ard/zdf-onlinestudie 2008 - internetverbreitung: Groesster zuwachs bei silver-surfern." [Online]. Available: http://www.ard-zdf-onlinestudie.de/fileadmin/Online08/Eimeren_I.pdf

[3] M. Fisch and C. Gscheidle, "Ergebnisse der ard/zdf-onlinestudie 2008 - mitmachnetz web 2.0: Rege beteiligung nur in communitys." [Online]. Available: http://www.ard-zdf-onlinestudie.de/fileadmin/Online08/Fisch_II.pdf

[4] N. online, "Buzzmetrics," May 2008. [Online]. Available: http://de.nielsen.com/products/documents/NielsenonlineBuzzMetrics20080521.pdf

[5] D. Gibson, K. Punera, and A. Tomkins, "The volume and evolution of web page templates," pp. 830–839, 2005.

[6] T. Gottron, "Content extraction: Identifying the main content in html documents," Ph.D. dissertation, Johannes-Gutenberg University Mainz, 2008. [Online]. Available: http://ubm.opus.hbz-nrw.de/volltexte/2009/1859/pdf/diss.pdf

[7] L. Yi, B. Liu, and X. Li, "Eliminating noisy information in web pages for data mining," pp. 296–305, 2003.

[8] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.

[9] B. Liu, R. Grossman, and Y. Zhai, "Mining data records in web pages," in *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2003, pp. 601–606.

[10] M. lvarez, A. Pan, J. Raposo, F. Bellas, and F. Cacheda, "Finding and extracting data records from web pages," *Journal of Signal Processing Systems*, 2008.

[11] Y. Zhai and B. Liu, "Web data extraction based on partial tree alignment," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*. New York, NY, USA: ACM, 2005, pp. 76–85.

[12] Y. Kim, J. Park, T. Kim, and J. Choi, "Web information extraction by html tree edit distance matching," in *ICCIT '07: Proceedings of the 2007 International Conference on Convergence Information Technology*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 2455–2460.

[13] B. Liu and Y. Zhai, "Net - a system for extracting web data from flat and nested data records," in *Proceedings of 6th International Conference on Web Information Systems Engineering (WISE-05*, 2005.

[14] S. P. Algur and P. S. Hiremath, "Extraction of flat and nested data records from web pages," in *AusDM '06: Proceedings of the fifth Australasian conference on Data mining and analytics*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 163–168.