Mining user-generated comments
Julien Subercaze, Christophe Gravier, Frederique Laforest

To cite this version:
Julien Subercaze, Christophe Gravier, Frederique Laforest. Mining user-generated comments. IEEE/WIC/ACM International Conference on Web Intelligence, Dec 2015, Singapour, Singapore. Proceedings of the IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) 2015. <hal-01187082>

HAL Id: hal-01187082
https://hal.archives-ouvertes.fr/hal-01187082
Submitted on 28 Aug 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Mining user-generated comments

Julien Subercaze, Christophe Gravier, Frédérique Laforest
Université de Lyon, F-42023, Saint-Etienne, France
CNRS, UMR5516, Laboratoire Hubert Curien, F-42000, Saint-Etienne, France
Université de Saint-Etienne, Jean Monnet, F-42000, Saint-Etienne, France

Abstract—Social-media websites, such as newspapers, blogs, and forums, are the main places of generation and exchange of user-generated comments. These comments are viable sources for opinion mining, descriptive annotations and information extraction. User-generated comments are formatted using a HTML template, they are therefore entwined with the other information in the HTML document. Their unsupervised extraction is thus a taxing issue – even greater when considering the extraction of nested answers by different users. This paper presents a novel technique (CommentsMiner) for unsupervised user comments extraction. Our approach uses both the theoretical framework of frequent subtree mining and data extraction techniques. We demonstrate that the comment mining task can be modelled as a constrained closed induced subtree mining problem followed by a learning-to-rank problem. Our experimental evaluations show that CommentsMiner solves the plain comments and nested comments extraction problems for 84% of a representative and accessible dataset, while outperforming existing baselines techniques.

I. INTRODUCTION

Possessing user-generated contents is one the great challenge in today’s Web ecosystem. Companies such as Twitter, Facebook, Instagram, Google, to name a few, have long understood the value of the content produced by their users. They managed to reach millions of users, mainly by offering free high-quality services. Analysing and processing these data raise many interesting research challenges. It is therefore not surprising that content posted on these mainstream platforms is attractive to researchers, especially because they are accessible as structured data through APIs. Although these social media and networks are in the limelight, they however represent only a fraction of user-generated content on the Web. Other user-generated content include reviews, comments, wikis and many other ways to create content on the Web.

This content is not centralized on mainstream platforms but is rather spread all over the Web, making it more difficult to reach out. Still, this user-generated content offers promising business opportunities. For instance, websites gathering user reviews on specific products receive a wide audience. User-generated comment business is, although less remarked than large social networks, one of the much competitive market in today’s Web. Many companies are engaged into the user-generated comments services: Disqus (founded 2007), LiveFyre (2009), Facebook comments plugin (2012), SolidOpinion (2013), Discourse (2013), to name a few. They all offer similar services: third party commenting frameworks for webmasters and blog owners. The ultimate goal being to enlarge their community of users and to let their user-generated content enter the company data silo.

Nonetheless, this does not address the problem of crawling and extracting user-generated comments at Web Scale. Gathering user-generated comments at Web Scale offer not only business opportunities but also research issues. That is the reason why Web content extraction and social media analysis has gained a lot of traction in the past years [1], [2]. Many major conferences have included mining and analysis tracks on social media content – this includes Intelligent Web Data Mining, Web Mining Applications, and Web Mining and Warehousing tracks in past WI conferences. However, the comment mining task, expected to be unsupervised for Web-scale extraction, is surprisingly understudied.

In this paper, we propose CommentsMiner, a two-stage algorithm that extracts comments from webpages along their conversational structure. Our approach allows nested comments extraction, which enables conversation extraction, a decisive feature for social analysis. Our scientific contributions are as following:

1) We notice that comments constitute frequent subtrees in the DOM. Thus, we demonstrate that comments extraction task can be modelled by utilizing frequent closed induced subtree mining along with a learning-to-rank model. Our approach differs from existing solutions insofar as we rely on a sound theoretical framework and benefits from the vast literature and algorithms on both frequent subtree mining and learning-to-rank.

2) We devise several constraints that drastically restrict the number of generated subtrees candidates, even under a small support value $s$ ($s \geq 2$) – these constraints include subtree positions in the DOM tree as well as textual features.

3) We demonstrate that our offline learning-to-rank approach utilizing densitometric features, achieves a perfect selection of the subtree pattern formatting comments for 84.37% of a representative and accessible dataset.

The entire process of CommentsMiner on data collected for the evaluation is done within 200 milliseconds in average per domain, which makes it a candidate for Web scale comments.

CommentsMiner achieves a perfect mining score on the TestBed for information extraction from Deep Web (TBDW)$^1$, and we present further results on a surrogate dataset called NUCE – another technical contribution presented with this paper, publicly available on its webpage$^2$.

---

$^1$http://daisen.cc.kyushu-u.ac.jp/TBDW/
$^2$http://datasets-satin.telecom-st-etteine.fr/cgravier/nuce/
Our goal is to find the subtree pattern $p$ in the HTML using a pattern that can be used for generating comments – we are mining the pattern $p$. Labels are omitted for readability.

**II. OVERVIEW OF CommentsMiner**

On the rendered webpage, user-generated comments are commonly positioned under the content they discuss. Under the hood, these comments are stored into a database on the server serving the content, or on the servers of third party providers (Disqus, Livefyre, ...).

When a page is viewed by a user, comments related to the page are loaded from the database and formatted into HTML using a pattern $p$. Comments are then assembled into a single HTML fragment with respect to the structure of the conversation. This HTML fragment is inserted into the webpage with a template mechanism as depicted in Figure 1, or using an iframe in the case of third-party services. From a datastructure point of view, the conversation made of comments is a tree. The root of the tree is the beginning of the comments section. This root is not visible as such on the webpage. Top level comments are children from the root, and their respective answers are commonly indented on the webpage. Top level comments are under the content they discuss. Under the set of vertices associating a HTML tag, $T$, $d$ is maximal if none of its supertrees is frequent, $C$. Gray-tinted subtrees constitute $C$. They are the occurences of the pattern $p$ used for generating comments – we are mining the pattern $p$. Labels are omitted for readability.

![CommentsMiner diagram](image)

**Fig. 1: Example of nested comments on www.reddit.com, in which $q = 5$. Gray-tinted subtrees constitute $C$. They are the occurrences of the pattern $p$ used for generating comments – we are mining the pattern $p$. Labels are omitted for readability.**
Candidate generation: Winner Selection:

set of candidates

pattern

winner 

..., ...

...

,...

,...

DOM tree

Constrained, frequent induced subtree mining

Rank candidates with features

Winner Selection:

Offline Learning

Ranking

Model

Fig. 2: Outline of CommentsMiner

DOM tree

Comments Extraction:

match pattern against DOM tree

Comments Extraction:

{ C₁, C₂, ..., Cₙ }

Conversation

rebuild conversation using parentship in DOM

{ C₁, C₂, ..., Cₙ }

Conversation

Fig. 3: Different types of subtree from tree (a) : bottom-up subtree (b) ; induced subtree (c) ; embedded subtree (d)

and closed if none of its supertrees has the same support [4].

Trees are categorized along three criteria: rooted/unrooted, ordered/unordered and labeled/unlabeled. A tree is called rooted if there exists a node that has been designated the root, in which case the tree may be traversed in two directions: towards and away from the root. A tree is said to be ordered if an ordering for the children of each vertex has been defined. Finally a tree is a labeled tree if each node is given a unique label. The interested reader is invited to refer to [5] for a detailed review on frequent subtree mining.

For our concern – subtree mining within DOM trees – we consider the case of rooted labeled ordered trees. There are different types of subtrees for there exists different types of trees. We consider a rooted tree T with vertex set V and edge set E and a subtree T' with respectively V' its vertex set and E' its edge set. In what follows, we present the three main types of subtrees with their definitions.

**Bottom-up subtree:** T' is a bottom-up subtree from T iff: V' ⊆ V, E' ⊆ E; for a vertex v ∈ V, if v ∈ V' then all descendants of v are in V'; the ordering of the siblings must be preserved in the subtree. Intuitively a bottom-up subtree T' can be obtained by taking a vertex from V together will all its descendants and the corresponding edges.

**Induced subtree:** T' is an induced subtree from T iff: V' ⊆ V, E' ⊆ E; for a vertex v ∈ V, the left-to-right ordering of the siblings is preserved in the subtree, i.e. it should be subordering of the corresponding vertices in T. An induced subtree T' can be obtained by repeatedly removing leaves.

**Embedded subtree:** T' is an embedded subtree from T iff: E' ⊆ E, (v₁, v₂) ∈ E' where v₁ is the parent of v₂ in T' only if v₁ is ancestor of v₂ in T. An embedded subtree T' must not break the parentship relations among the vertices.

Figure 3 illustrates these three types of subtrees from a given data tree through three different examples on the same data tree 3 (a).

To better understand the difference between the subtrees, one can say that bottom-up subtrees are complete subtrees while induced subtrees allow to remove nodes horizontally in the subtree and finally embedded subtrees allow both horizontal and vertical removals. We have the following relationship: bottom-up subtree ⊆ induced subtree ⊆ embedded subtree.

Algorithms’ complexities vary with the nature of the data tree and with the type of subtree one wants to extract. Mining bottom-up subtrees is faster than mining induced subtrees which is itself than mining embedded subtrees. The time complexity is linked to the number of subtrees for each type and to the complexity of the algorithms and data structures. The impact on performance being serious [5], it is therefore highly desirable to identify precisely which type of subtrees we want to extract. We discuss this issue, as well as performance optimizations in the next subsections.
B. Application to comments extraction

Figure 4 describes an example of DOM fragment containing, (a), a single comment, and (b), the same comment with an answer. This simplified example is used as an illustration for discussions in this section. In this figure, the pattern we should identify is \((A, P)\)DIV.

![Fig. 4: Example of subtrees. The comment pattern is \((A, P)\)DIV (in Newick tree format).](image)

In our running example, the answer is inserted in the middle of the original comment between the tags \(A\) and \(P\). This situation is relatively common, we also encountered several cases where the answer is inserted at the end of the comment – in this example it would be after the \(P\) in the comment.

In order to mine comments that are nested in a conversational structure, such as in Figure 4, the subtree mining process must be able to skip leaves horizontally. Thus, using bottom-up subtrees would not be sufficient to match this pattern. However the vertical removal that is permitted using embedded subtrees is not a wishful feature and would lead to unnecessary expensive computations. The frequent subtrees we are mining have therefore the property to be induced.

In user-generated comments, the content of comments itself may contain not only user-generated text but also HTML tags. These tags are at the discretion of the users. For example some websites allow users to format their text with a limited whitelist of tags such as `<b>`, `<i>`, `<em>` etc. The original template subtree can then be extended by other tags – this is exemplified by blank leaves under gray-tinted subtrees \(c_t \in \mathcal{C}\) in Figure 1. During this mining step, it is not possible to identify the target subtree from its supertrees that are due to user formatting. Thus, maximal subtree mining is too restrictive. However closed subtree mining matches our requirement, since it will output both the target and its supertrees: in the case where the comments have different formatings, the target subtree will have a greater support than its supertrees. On the other hand, when all comments share the same formatting, the target pattern is extended to the formatted one, thus allowing the extraction of the comments. Thus, the frequent induced subtrees that we are mining have the second property to be closed.

Intuitively, HTML attributes may be prone to help in frequent subtree mining for the problem of comments extraction. However, within a single webpage, attributes may be highly variable for various reasons, for instance: comments color alternative, highlighted or downvoted comments, internal comments ID etc. Frequent attributed subtree mining \[6\] would have resulted in a loss of generality. We discarded this approach for the global process, while retaining comments characteristics as an optimization to drive the mining process when possible (See III-D).

For our implementation, we use CMTTreeMiner \[4\] as the algorithm of our choice – because it is the only one providing frequent subtree mining of ordered closed subtrees as reported in \[7\]. We adapted the algorithm to take into account the constraints described in the next section.

C. Constraints

The search space of a DOM tree is very large, as reported by HTTP archive\(^3\), an average DOM tree contains 1300 nodes. Regarding the performance reported by the authors of CMTTreeMiner, mining a tree of thousand nodes with a support of 2 (minimal number of comments to be frequent) could take minutes, even hours. To make this process tractable at large scale, we introduce constraints, that restrict the search space while guaranteeing that the target pattern will be present in the set of candidates. We define three domain-specific constraints to accomplish such task.

**Lowest common ancestor similarity.** As comments are located in a unique area of the DOM tree, occurrences of the target pattern are relatively close to each other. The tree distance between root occurrences of the target pattern is not a priori known, and may vary from page to page. However, the two root occurrences of the target pattern are in the same subbranch of the top tag, i.e. the `<body>` tag. Formally, the lowest common ancestor between two occurrences of the pattern cannot be the root of the DOM tree. This constraint reduces drastically the search space of CMTTreeMiner. At each step the subtrees of depth \(k\) are expanded. For each supertree resulting of this expansion, we compute a binary similarity matrix between its root occurrences – then we aimed at exploiting co-occurrences information for leveraging pattern candidates \[8\]. The similarity is equal to one if the lowest common ancestor of two occurrences is the root of the tree, zero otherwise. Using this distance we group occurrences into clusters and split the existing supertree into several supertrees. The binary similarity has the nice properties of being commutative and transitive, therefore it is easily computed in \(O(n)\), where \(n\) is the number of root occurrences.

**Blank occurrences deletion.** Another simple, yet very efficient constraint is based on the text associated to the occurrences. We discard patterns whose occurrences contains no text or identical text.

**Root and rightmost occurrences equality.** In CMTTreeMiner – also in \[9\] – induced subtrees occurrences are identified during the mining process using their rightmost occurrences. We denote \(RootOCC_t \) and \(RightOCC_t\) the sets of root and rightmost occurrences of a frequent subtree \(t\) in a datatree \(T\). Each comment has its own root and right most occurrence – they are not shared with other comments. The verification for any candidate subtree is therefore carried out with :

\[^3\]http://www.httparchive.org/trends.php?s=Top1000
\(|\text{RootOcc}_{i,T}| = |\text{RmoOcc}_{i,T}|\). This constraint does not limit
the search space, but drastically reduces the size of the output
set of candidates.

D. Optimizations

Optimizations, at the opposite to the above defined con-
straints, are not guaranteed to work in every case. Yet, for a
large number of encountered situations, they provide a notice-
able performance gain. The idea behind these optimizations
is to speed-up the mining process wherever possible while
still being able to complete the mining process when the
optimizations do not kick in.

In addition to the constraints, we introduce two optimisation
in order to speed up the first step of our system. De-
creasing the support \(s\) makes the number of pattern candidates
exponentially grows, hence the computational complexity of
any comment extractor. Increasing \(s\) will limit the number
of generated subtrees, but increase the probability to miss
the expected pattern. We therefore choose \(s = 2\), inspired
by the literature review [10]. This is the practical accepted
value for most similar extractors. Using such a low value for
\(s\) is computationally expensive, since the number of frequent
subtrees matching this support value is very high in a DOM
page. We therefore introduce an optimization called Attributes
Fallback to reduce the number of initial nodes that will form
the pattern: we initialize the first mining steps with nodes that
only have attributes containing words related to comments.
Words such as comments, reaction, posts . . . are looked for.
When no such initial nodes are to be found, the algorithm falls
back to its standard version.

The second optimisation is Page Optimization: starting
from the leaves of the DOM tree, we delete leaves containing
no text. This operation is repeated until a fix point is reached.
This considerably reduces the size of the search space. If the
pattern was mined using this preprocessing step, the extraction
must also run this preprocessing step.

The impact of the optimizations and their combinations,
depending on the minimum support is presented in Figure 5.
Combining the two optimizations reduce the number of sub-
trees by a factor 4 on our dataset\(^4\) for small minimum support
values, which is meaningful for the task of comments extrac-
tion. The memory footprint enhancement is strongly linearly
correlated to this factor. Evaluation on the dataset showed that
the winner candidate was not removed when only nodes with
attributes related to comments were used for initialization.

E. Output

This stage of the algorithm outputs a set of candidate
patterns, among which include the pattern used to generate
the comments on the Webpage. The set of candidate patterns
is a subset of all generated patterns as seen in Figure 5. To
be a candidate for the next stage, a pattern must validate the
constraints defined in Section III-C. In the practice, the number
of candidate patterns rarely exceeds twenty.

\(^4\)This dataset is presented and discussed in Section VI-B

Fig. 5: Influence of the optimizations on the number of
generated patterns with respect to the minimum support value.

IV. Winner Selection

For a given webpage, the previous subtree mining step out-
puts a set of candidate patterns. Among these candidates, we
aim now at finding the pattern \(p\), that was used as the template.
Finding the pattern \(p\), that was used as the template to embed
comments, among the subtrees issued by the previous stage
can be seen as a ranking problem, where only the first rank
matters. To rank these candidates patterns, we use textual and
densitometric features as input to learning to rank algorithms,
including: SVMRank, MART, RankNet, RankBoost, AdaRank,
Random Forests, and Genetic Programming. We first describe
the features, then the ranking measure. Experimental results
are presented in Section VI.

A. Features description

The main characteristics that distinguish candidates for
ranking are both text and densitometric features. One can
observe that user-generated content is of variable length [11]
– unlike menus for which the length and the number of words
are very similar among menu items. It is also usually forbidden
to include links in comments to avoid spamming, we therefore
expect a low density of link in the HTML code. The text
density (ratio text vs code) of user-generated comments is
also significantly different from the one of boilerplate [12].
Therefore we exploit these characteristics as a set of eight
features (listed in Table I). As the content of user-generated
comments differ significantly, the average and the standard
deviation of each of these features convey the heterogeneity
between occurrences of the same candidate subtree.

B. Ranking measures

Our work deals with a special case of learning to rank,
where only the most relevant candidate matters, regardless
of other candidates. This kind of binary relevance is usually
denoted as Winner Takes All (WTA) [13]. Assume that we are
given the set of candidates pattern \(P := \{p_1, . . . , p_l\}\) along
with their label \(y \in \{0, 1\}\), such as \(y_i\) is the binary relevance
of the \(i\)-th candidate pattern. This relevance is a score of one
In this section we first present the baselines and the experimental setup. Then we report and discuss the accuracy and performance of CommentsMiner for the comments and their conversational structure extraction task.

A. Baselines

To the best of our knowledge, only MiBat [14] was designed for the comment extraction task. Unfortunately, the materials used in MiBat (software or datasets) are not publicly available, nor upon request. The perfect matching success rate of 75.653% was obtained for several pages belonging to the same Web domain (this is inferred from the illustrations within the paper, yet the precise number is unknown) – this skews the evaluation. Another baseline is DEPTA, a follow-up of MDR (see Section VII). DEPTA requires a full browser rendering and a visual analytics that result in poor scalability, and it is unable to extract parent-child relationships. While DEPTA is not accessible, MDR can be retrieved online\(^6\) – which makes it the candidate to be considered a standard baseline in several works as reported in the survey \([10]\).

Other eligible candidates are TPC and RST, yet none are publicly available. They were however evaluated against the same dataset, the TBDW dataset. CommentsMiner achieves a success rate of 100% on this dataset, which makes hardly a difference with TPC and RST (resp. 96.23% and 98.06% precision, and resp. 97.03% and 97.88% recall value). Henceforth, we will focus on the more challenging dataset that is NUCE.

B. Datasets

Both TPC [15] and RST [16] are competitors to our approach. They were evaluated using the TestBed for information extraction from Deep Web (TBDW)\(^6\). We discuss how CommentsMiner performs on this dataset with respect to these competitors in the next section (VI-A). However, there are some primary issues on benchmarking the comments and nested comments extraction task on the TBDW dataset – mainly, it no longer reflect today’s Web programming habits. Particularly:

- Most of the data to extract in this dataset are within "table" and "form" tags – it was a common practice in 2003 which has now completely vanished.
- Today’s Web pages are more complex, which results in a significantly increased search space. According to http://httparchive.org/, the average webpage size has increased by 237% from December 2010\(^7\) to February 2015\(^8\). However, this website does not provide figures before 2010, but according to the same figure in 2003 reported in [17], an estimation of the increase between early 2003 and fall 2013 is 1,723%.
- In addition, there is no case of nested subregions within this dataset, which makes it difficult to evaluate the multi-level nesting extraction.

\(^5\)http://httparchive.org/interesting.php?l=Dec%2028%202010
\(^6\)http://daisen.cc.kyushu-u.ac.jp/TBDW/
\(^7\)http://httparchive.org/interesting.php?l=Dec%2028%202010
\(^8\)http://httparchive.org/interesting.php?l=Dec%2015%202013

<table>
<thead>
<tr>
<th>Observation</th>
<th>Description</th>
<th>Average-based features</th>
<th>STD-based features</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextLength</td>
<td>Text length of all instances for a pattern candidate.</td>
<td>TextLengthAVG</td>
<td>TextLengthSTD</td>
</tr>
<tr>
<td>LinkDensity</td>
<td>Percentage of characters between &lt;a&gt; tags.</td>
<td>LinkDensityAVG</td>
<td>LinkDensitySTD</td>
</tr>
<tr>
<td>TextDensity</td>
<td>Ratio of characters that are not part of HTML boilerplate over all characters</td>
<td>TextLengthAVG</td>
<td>TextLengthSTD</td>
</tr>
<tr>
<td>WordsVolume</td>
<td>Number of words, excluding HTML tags and attributes.</td>
<td>WordsVolumeAVG</td>
<td>WordsVolumeSTD</td>
</tr>
</tbody>
</table>

TABLE I: Features are computed over all instances of each pattern and are two-fold: one for its average value, and one for its standard deviation (STD) value.

\(WTA(P, \hat{g}) = \begin{cases} 
1 & \text{iff } \forall i \in \{1, \ldots, |P|\}, \hat{g}(p_i) = y_i \\
0 & \text{otherwise.} 
\end{cases} \)  

(1)

Note that CommentsMiner relies on a ranking function that must be learnt. However, CommentsMiner is considered unsupervised: once the ranking function is learnt, it can be reused for unknown Web domains, and without further learning. This is consistent with the classification on this criterion introduced by the recent and exhaustive literature review provided in [10].
TABLE II: Performance and settings of trained learners on the NUCE dataset for extracting the exact pattern $p$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Settings</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Type</td>
<td>Best model settings</td>
</tr>
<tr>
<td>MART</td>
<td>Pointwise</td>
<td>1,000 trees</td>
</tr>
<tr>
<td>SVMRank</td>
<td>Pairwise</td>
<td>$RBF$, $c = 0.1$</td>
</tr>
<tr>
<td>RankNet</td>
<td>Pairwise</td>
<td>100 iterations</td>
</tr>
<tr>
<td>RankBoost</td>
<td>Pairwise</td>
<td>300 rounds</td>
</tr>
<tr>
<td>AdaRank</td>
<td>Listwise</td>
<td>WTA for training</td>
</tr>
<tr>
<td>Coord. Ascent</td>
<td>Listwise</td>
<td>WTA for training</td>
</tr>
<tr>
<td>ListNet</td>
<td>Listwise</td>
<td>1,500 iterations</td>
</tr>
<tr>
<td>Gen. Prog.</td>
<td>Listwise</td>
<td>50 iterations</td>
</tr>
</tbody>
</table>

The dataset is dedicated to Web search results for 51 different search engines. No webpage include comments to extract – it is dedicated for the extraction of the frequent subtrees that represents web search results. This is still of interest since the problem of comments extraction can be neatly addressed using densitometrics features that may not apply on other use cases.

Since CommentsMiner achieve a perfect score for frequent subtree mining for the ground truth offered by the TBDW dataset, we built a more challenging dataset with attributes including: i) up-to-date web programming paradigms, ii) diverse and multilingual web domains, and iii) Webpages with nested regions. We proceeded as follows: i) find relevant domains starting from Google News in English, French and German. When we ran out of Web domains on recent news, we search new domains from Reddit, Fark and Metafilter, then ii) for each domain, find a page containing more than two comments and download its content through the web browser in order to avoid AJAX calls issue [18]. We strictly consider only one page per domain. Some services like Wordpress, Disqus, Livefyre, Facebook, etc., provide commenting features. In order to avoid any bias, we kept one page using each service. The dataset consists in 211 labeled Web pages. We called this labeled dataset NUCE, which stands for Nested User-generated Content Extraction dataset. Our surrogate dataset is publicly available to download\footnote{http://datasets-satin.telecom-st-etienne.fr/cgravier/nuce/}, and includes for each page its browser-side rendered webpage as well as the associated ground truth – the subtree pattern that a comment extraction algorithm is expected to mine.

C. Results

The evaluation depends on the quality of the learning-to-rank step since the expected pattern $p$ is always included in the set of the pattern candidates set (as discussed in Section III). We utilized different learning methods for learning to rank pattern candidates. While it is out of the scope of this paper to provide a complete state-of-the-art on learning-to-rank methods, the authors can refer to [19] and [20] for further details. The genetic programs were trained using the WTA metric and the following operators were available for the learner: addition, multiplication, subtraction, division, power, along with any values in the range $[2; 10,000]$. All learners were trained and tested using the eight features described in Table I. Results are presented in Table II. Training was done using 20% data partitioning and a five-fold cross-validation.

ListNet best model provides a P@1 of 90.170 over 100 runs. However ListNet-based learning-to-rank models suffer from a very significant standard deviation. We conclude that Genetic Programming models are the most suitable for the learning-to-rank step. Although those models do not achieve the best success rate ($84.375$ in average), it is still very good while providing more guarantees on its generalization. Genetic Programming based models offers a standard deviation of $0.854$. Genetic Programming models therefore offer a stability of success rates of the utmost practical interest for a learn once, extract many crawling strategy.

The expected patterns to mine exhibits different depths and sizes, as reported in Figure III.

D. Performance and Scalability

The experiments were run on a 64-bit Linux server using an Intel Xeon E5430 and 4 Gb of RAM. Using this commodity hardware, the wall-clock time of the two-step algorithm of CommentsMiner is 201 ms in average. The execution time presents a significant standard deviation (242 ms) due to the difference in page sizes (hence DOM tree density), which is $149$ Kb in average, but with a standard deviation of $105$ Kb. This impacts the pattern size in number of nodes as well ($14.10$ nodes in average with a standard deviation of $10.8$). As an illustration of this performance – given $500$ US$ and the cost of a m3.large server at Amazon EC2 ($0.140$ US$ per hour at the time of writing) – it is therefore possible to process $63,958,209$ web pages with just $500$ US$. Each Web page can also be processed in isolation, which provides another scalability improvement.

VII. RELATED WORKS

DEPTA [21] – an extension of the work reported in [22] – first processes the page using a Web browser in order to get the boundaries information of each DOM node and later detects nested rectangles – thus building a tag tree where the parent relationship indicates a containment in the rendered page. DEPTA utilizes a string edit distance to cluster similar nodes into regions – a similar technique used by [23], while replacing the tree edit distance with a token edit distance. MiBAT [14] – an automatic extraction framework of Web data record containing user-generated content – relies on domain
constraints to acquire anchor points information. For example, each forum post must have exactly one tree containing “post-date”. Despite their novel contributions, these works are unable to provide a solution for classifying frequent regions with respect to Web document structure [24]. Moreover, none are able to mine nested frequent subtrees, i.e. comments published in answer to other comments and situated in a subtree of the first level comment subtree.

Other works focus on devising domain-agnostic region extractors. [10] provides a contemporary and exhaustive analysis of more than a hundred of existing region extractors, including a nesting level handling criterion. Among the 14 analyzed works for region extraction, only three proposals (Tag Path Clustering (TPC [15]), Vision-based Page Segmentation (VIPS [25]), and Record Segmentation Tree (RST [16]) are able to maintain a relationship between the subregions (referred as “multi-level nesting” in this survey). This feature denotes the ability to identify nested sub-regions within a region that allows the extraction of the conversational structure of the comments. Among these proposals, VIPS outputs the segmentation of the webpage in regions, without providing any information on the relative relevance of the subregion. Therefore, it is not suitable to use VIPS for comments extraction – as no selection scheme provides the probability distribution for comments in regions.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we presented CommentsMiner, a novel approach to extract user-generated comments. CommentsMiner bridges the gap between frequent subtree mining and web information extraction by successfully extracting HTML templates that embed user-generated comments. A specificity of users comments is their conversational structure. Our approach based on constrained mining of closed frequent subtrees is able to extract nested comments. By constraining the mining process, we are able to avoid the combinatorial explosion that usually characterizes subtree mining. To identify the winner subtree among those output by the mining step, we use a learning-to-rank approach and compared the result of several algorithms. We finally compare our extraction result to existing approaches on both a popular and a surrogate datasets, thus acknowledging the improvement brought by CommentsMiner on the comment mining task.

ACKNOWLEDGEMENT

This work was partially funded by the project grant Greenfeed from the French FUI (Unique Interministerial Fund), and the project grant FIRE from the French Fund for a digital society (FSN).

REFERENCES