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 `CommentsMiner`, a novel technique for unsupervised users 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 baseline techniques.**

I. Introduction

Possessing user-generated contents is one of 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 are attractive to researchers, especially because they are accessible as structured data through APIs. Although these social media and networks are in the limelight, they 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 comments business is, altough it stands in the shadow of the large social networks, one of the much competitive markets 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 is 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 offers 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 comments 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 with 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 show that comments constitute frequent subtrees in the DOM. Thus, we demonstrate that the 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 subtree 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 available 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 extraction. `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 – a technical contribution presented with this paper, and publicly available on its webpage[2].

[1] `http://daisen.cc.kyushu-u.ac.jp/TBDW/`

[2] `http://datasets-satin.telecom-st-etienne.fr/cgravier/nuce/`

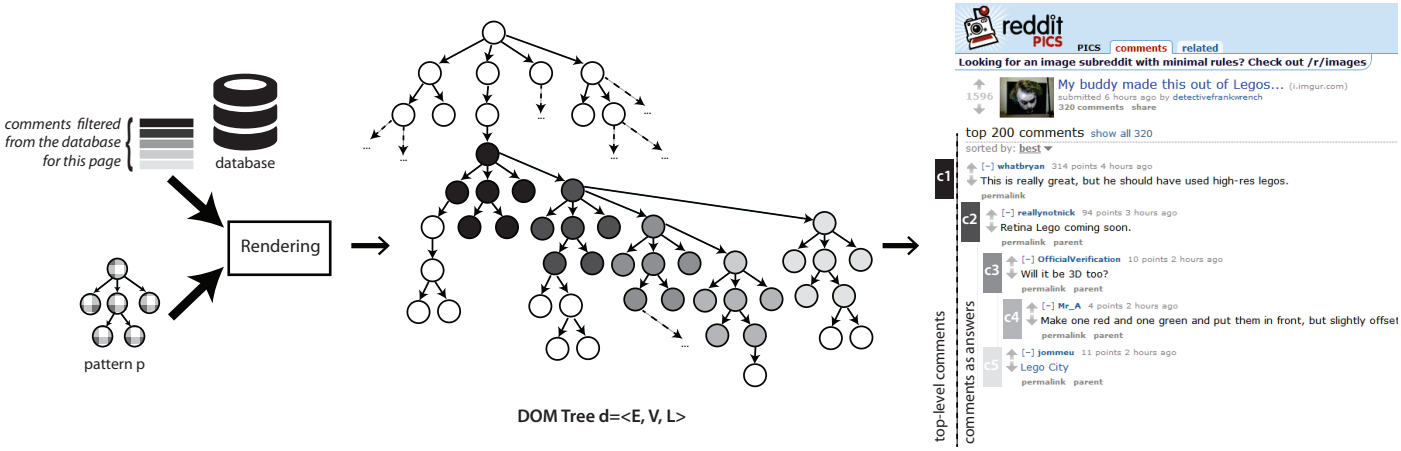

Fig. 1: Example of nested comments on `www.reddit.com`, in which $q = 5$. Gray-tinted subtrees constitute $\mathcal{C}$. They are the occurences of the pattern p used for generating comments – we are mining the pattern p. Labels are ommitted for readability.

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 provider (Disqus, Livefyre, ...). We assume here that we do not have access to these databases.

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 this root, and their respective answers are usually indented to the right for simple visualization.

The problem of users' comments extraction can be defined as retrieving the root node of the comments and the pattern p that was used to generate the HTML fragments embedding users' comments. This pattern is valid at site scale. As a consequence, once the pattern is determined, it can be used to extract comments from every page of the site.

More formally, a Web document d is a rooted, directed, and ordered tree (the DOM tree) noted $d = \langle E, V, T \rangle$ where T is the set of HTML tags, E the set of DOM nodes, and V the set of vertices associating a HTML tag $t \in T$ to any $e \in E$. Our goal is to find the subtree pattern $p = \langle E', V', T' \rangle$, with $V' \subset V,\ E' \subset E,\ T' \subset T$, such as $\mathcal{O}_p$, the set of occurences of the pattern p in the DOM tree validates :

- $\forall c \in \mathcal{O}_p, c$ is a DOM fragment containing a user-generated comment.
- $|\mathcal{O}_p| = q, q \geq 2$ where q is the actual number of comments in d.

We aim at automatically identifying the pattern p for any given webpage d, without any a priori knowledge about the structure of d. Whenever a webpage contains at least two comments, the pattern used to render comments is repeated within the page. Therefore, the pattern identification task can be described as a frequent subtree mining task. Figure 2 depicts the overall architecture of `CommentsMiner`. `CommentsMiner` works in two stages.

First, we consider the problem of comments and nested comments mining as constrained frequent subtree mining. We generate the set of subtrees that are candidate for the comment extraction task. `CommentsMiner` uses domain-specific constraints to reduce the number of candidate subtrees. This stage is detailed in the next section.

Second, we learn a ranking model to select the winner subtree among the candidates. An important remark is that our ranking model is learnt using the content of the comments – i.e. textual features – and not using any tree related feature. Consequently, the model is agnostic to the structure of training pages, and generalizes very well on unknown pages. This process is detailed in Section IV.

Once the pattern has been identified, the system can perform the extraction of the user-generated comments (Section V). Section VI presents our experimental evaluation and discusses the performance and the scalability of `CommentsMiner`.

III. Candidates Generation

We first recall some notions on frequent subtree mining. A reader familiar with this topic can skip this subsection and head directly to Subsection III-B, that details the application of frequent subtree mining to comments extraction.

A. Frequent Subtree Mining

Tree mining deals with the task of finding patterns of interest in a single tree or in a forest [3]. The type of patterns to be extracted, as well as the trees they should be

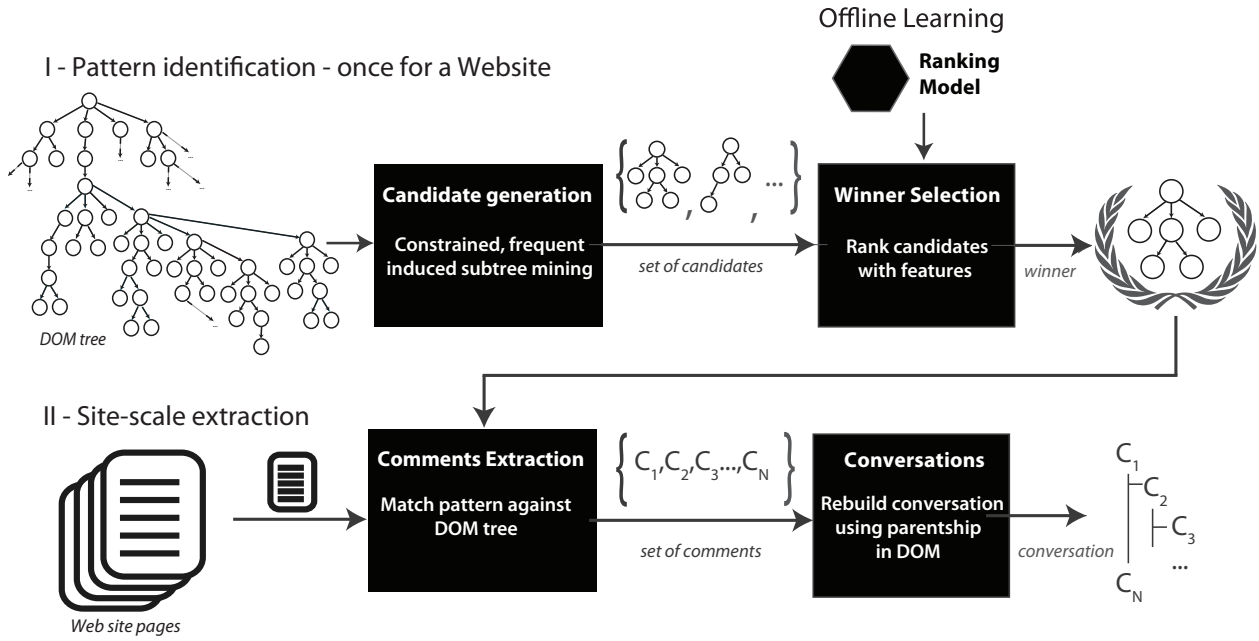

Fig. 2: Outline of CommentsMiner

extracted from, may differ in nature. We briefly recall that a subtree is frequent if its support is greater than a minimum support s. A subtree t is maximal if none of its supertrees is frequent. It is 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. In what follows, we consider a rooted tree $T = \langle E, V, T \rangle$ and a subtree $T' = \langle E', V', T' \rangle$ and present the three main types of subtrees. Figure 3 illustrates them on a given data tree.

Bottom-up subtree: T' is a *bottom-up subtree* from T iff: $V' \subseteq V$, $E' \subseteq E$; for a vertex $v \in V$, if $v \in V'$ 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: $V' \subseteq V$, $E' \subseteq E$; for a vertex $v \in V$, the left-to-right ordering of the siblings are 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: $E' \subseteq E$, $(v_1, v_2) \in E'$ where v_1 is the parent of v_2 in T' only if v_1 is ancestor of v_2 in T. An embedded subtree T' must not break the parentship relations among the vertices.

To better understand the differences between these subtrees, one can say that bottom-up subtrees are *complete* subtrees while inducted 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 $\subseteq$ induced subtree $\subseteq$ embedded subtree.

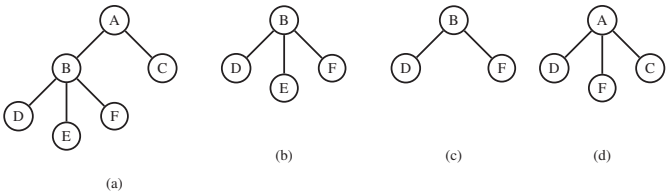

Fig. 3: Different types of subtree from tree (a) : bottom-up subtree (b) ; induced subtree (c) ; embedded subtree (d)

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 faster 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 important [5], it is therefore highly desirable to identify precisely the 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 (in Newick tree format[3]).

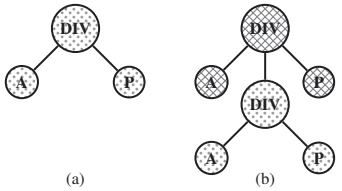

Fig. 4: Two examples of two nested comment subtrees with the comment pattern (A,P)DIV.

Figure 4(b) depicts the typical conversational structure of comments in a Webpage. An answer to a comment is a child of this comment. There are no rules about where the root of an answer is inserted within the original comment – as it can be anywhere depending on the website template. We encountered various situations during our research. 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. In Figure 4(b), the system must be able to horizontally split the first (A,P)DIV pattern occurence in order to exclude the DIV child node as it is the root of the second occurence of the (A,P)DIV pattern that we would mine in this example. 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 necessary feature and would lead to necessary expensive computation overhead. The frequent subtrees we are mining are therefore induced subtrees.

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>, etc. The original template subtree can then be extended by other tags – this is exemplified by blank leaves under gray-tinted subtrees $c_i \in \mathcal{C}$ 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 formattings, 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. We mine closed induced frequent subtrees.

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 a possible optimization to drive the mining process (see III-D).

For our implementation, we use CMTreeMiner [4] as the algorithm of our choice – because it is the only one providing frequent subtree mining of ordered induced closed subtrees as reported in [7]. We have 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[4], an average DOM tree contains 1300 nodes. Regarding the performance reported by the authors of CMTreeMiner, 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 this 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 CMTreeMiner. At each iteration 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-occurences information for leveraging pattern candidates [8]. The similarity is equal to one if the *lowest common ancestor* of two occurences 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 occurences 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. The bottom-up traversal of the tree by CMTreeMiner makes it possible.

Root and rightmost occurrences equality. In CMTreeMiner – also in [9] – induced subtrees occurrences are identified during the mining process using their rightmost occurrences. We denote $RootOcc_{t,T}$ and $RmoOcc_{t,T}$ 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

[3] http://evolution.genetics.washington.edu/phylip/newicktree.html

[4] http://www.httparchive.org/trends.php?s=Top1000

– they are not shared with other comments. The verification for any candidate subtree is therefore carried out with: $|RootOcc_{t,T}| = |RmoOcc_{t,T}|$. 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 constraints, are not guaranteed to work in every case. Yet, for a large number of encountered situations, they provide a noticeable performance gain. This is especially true for small support values, which is part of the research objectives. The idea behind these optimizations is to speed-up the mining process wherever possible, while still being to able to complete the mining process when these optimizations do not kick in.

In addition to the constraints, we introduce two optimisations in order to speed up the first step of our system. Decreasing the support s makes the number of pattern candidates exponentially grow, hence the computational complexity of any comment extraction. Increasing s limits the number of generated subtrees, but increases 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 found, the algorithm falls back to its standard version.

The second optimisation is *Page Optimization*: starting from the leafs of the DOM tree, we delete leafs 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 reduces the number of candidate subtrees by a factor 4 on our dataset[5] for small minimum support values, which is meaningful for the task of comments extraction. The memory footprint enhancement is strongly linearly correlated to this factor.

E. Output

This stage of the algorithm outputs a set of candidate patterns. 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 practice, the number of candidate patterns rarely exceeds twenty.

[5]This dataset is presented and discussed in Section VI-B

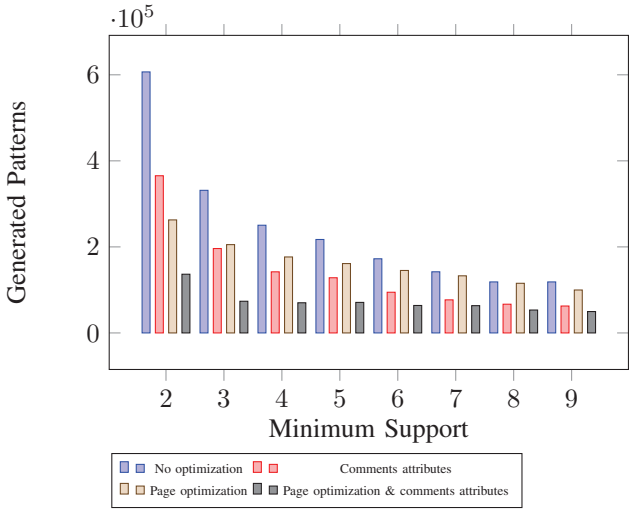

Fig. 5: Influence of the optimizations on the number of generated patterns with respect to the minimum support value.

IV. Winner Selection

For a given webpage, the previous subtree mining step outputs a set of candidate patterns. Among these candidates, we aim now at selecting the pattern p, that was used as the template to embed comments in the Web page. Finding this pattern p among the subtrees issued by the previous stage can be seen as a ranking problem, where only the first rank matters. To rank these candidate 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 $P := \{p_1, \ldots, p_l\}$ along

Observation	Description	Average-based features	STD-based features
TextLength	Text length of all instances for a pattern candidate.	`TextLengthAVG`	`TextLengthSTD`
LinkDensity	Percentage of characters between `<a></a>` tags.	`LinkDensityAVG`	`LinkDensitySTD`
TextDensity	Ratio of characters that are not part of HTML boilerplate over all characters (text and HTML tags).	`TextLengthAVG`	`TextLengthSTD`
WordsVolume	Number of words, excluding HTML tags and attributes.	`WordsVolumeAVG`	`WordsVolumeSTD`

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.

with their label $y_i \in \{0,1\}, i \in [0,l]$, such that y_i is the binary relevance of the i-th candidate pattern. This relevance is a score of one for the winner pattern, and zero otherwise. We also assume $\hat{g}(p_i)$ being the scoring function – estimating the rank of p_i belongs to $\{1,\ldots,|P|\}$. Given the set P and the function $\hat{g}$, WTA is defined as follows:

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

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].

V. Extraction

Once the winner pattern has been selected by the ranking process, we proceed to the extraction of the comments. As depicted in Figure 2, we first match the pattern against the DOM tree and then rebuild the conversational structure of the comments. Matching the pattern against the DOM tree is performed in linear time using a depth-first strategy (breadth-first is also suitable here): the algorithm first lists all the occurrences in the DOM tree of the root element of the pattern. For each occurrence, it successively checks that the first child of the pattern is matched in the datatree. The validation continues similarly to a depth-first tree traversal.

It may happen in some edge cases that the pattern is sufficiently common so that it is matched on other parts of the DOM than the comments. To avoid such a case, we group the occurrences that are close in the DOM using the same technique as the constraint of placement in the DOM. Afterwards, we use the parent relation between root elements of the occurrences to determine which comment answers another one. We start with the comments that have the lowest level in tree (highest depth). For each comment we traverse the tree towards the root and check if the encountered nodes are root nodes of a comment.

VI. Experimental results

In this section we first present the baseline and the experimental setup. Then we report and discuss the accuracy and performance of `CommentsMiner`.

A. Baselines

To the best of our knowledge, only MiBat [14] was specifically 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 as 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 a more challenging dataset : 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)[7]. 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 reflects 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[8] to February 2015[9]. 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 2003 and 2013 is 1,723%.

[6] `http://www.cs.uic.edu/~liub/WebDataExtraction/MDR-download.html`

[7] `http://daisen.cc.kyushu-u.ac.jp/TBDW/`

[8] `http://httparchive.org/interesting.php?l=Dec%2028%202010`

[9] `http://httparchive.org/interesting.php?l=Dec%2015%202013`

Algorithm		Settings	Results	
Name	*Type*	*Best model settings*	μ	σ
MART	Pointwise	$1,000$ *trees*	66.402	6.391
SVMRank	Pairwise	$RBF,\ c = 0.1$	77.145	6.397
RankNet	Pairwise	100 *iterations*	67.195	6.8204
RankBoost	Pairwise	300 *rounds*	84.027	4.9706
AdaRank	Listwise	WTA for training	66.041	15.423
Coord. Ascent	Listwise	WTA for training	81.619	3.642
ListNet	Listwise	$1,500$ *iterations*	90.170	13.521
Gen. Prog.	**Listwise**	**50 *iterations***	**84.375**	**0.854**
MiBAT (*baseline*)	Anchor Trees	N/A	75.653	Unknown

TABLE II: Performance and settings of trained learners on the NUCE dataset for extracting the exact pattern p.

- In addition, there is no case of nested subregions within this dataset, which makes it difficult to evaluate the multi-level nesting extraction.
- 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` achieves a perfect score for frequent subtree mining for the ground truth offered by the TBDW dataset, we built a more challenging dataset including: i) up-to-date web programming paradigms, ii) diverse and multilingual web domains, and iii) Webpages with nested regions. We proceeded as follows to create this dataset: 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 stricly 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 manually 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[10], 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, substraction, 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), they provide stronger guarantees on their generalization. Genetic programming models exhibit the lowest standard deviation (0.854). Genetic programming models therefore offer a stability of success rates of the utmost pratical interest for a *learn once, extract many* crawling strategy.

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

	Flat		Nested		Total	
	μ_f	σ_f	μ_n	σ_n	μ_t	σ_t
Pattern depth	4.24	1.79	5.69	2.74	4.58	2.1
Pattern size (#nodes)	12.8	9.83	18.39	12.91	14.1	10.8
Page size (Kb)	134	104	198	92	149	95

TABLE III: Winner patterns summary. Two classes of comments are observed – the first where no nested comments are present (flat), the second with nested comments.

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 is 201 ms in average. It 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 pages can also be processed in isolation, which provides another scalability factor.

VII. Related work

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

[10] `http://datasets-satin.telecom-st-etienne.fr/cgravier/nuce/`

constraints to acquire anchor points information. For example, each forum post must have exactly one tree containing "post-date". 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 work 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 classification of the subregion as comments, menus, ads, ... Therefore, it is not suitable to use VIPS for comments extraction.

VIII. Conclusions and future work

In this paper, we presented `CommentsMiner`, a novel approach to extract conversations of user-generated comments. `CommentsMiner` bridges the gap between frequent subtree mining and web information extraction by succesfully 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 dataset, thus acknowledging the improvement brought by `CommentsMiner` on the comment mining task.

Acknowledgement

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

References

[1] B. Liu and L. Zhang, "A survey of opinion mining and sentiment analysis," in *Mining text data*. Springer, 2012, pp. 415–463.

[2] E. Ferrara, P. De Meo, G. Fiumara, and R. Baumgartner, "Web data extraction, applications and techniques: A survey," *Knowledge-Based Systems*, vol. 70, pp. 301–323, 2014.

[3] M. Zaki, "Efficiently mining frequent trees in a forest: Algorithms and applications," *IEEE trans. on Knowledge and Data Engineering*, vol. 17, no. 8, pp. 1021–1035, 2005.

[4] Y. Chi, Y. Xia, Y. Yang, and R. R. Muntz, "Mining closed and maximal frequent subtrees from databases of labeled rooted trees," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 2, pp. 190–202, Feb 2005.

[5] Y. Chi, R. Muntz, S. Nijssen, and J. Kok, "Frequent subtree mining-an overview," *Fundamenta Informaticae*, vol. 66, no. 1-2, p. 161, 2005.

[6] A. Termier, M.-C. Rousset, M. Sebag, K. Ohara, T. Washio, and H. Motoda, "Dryadeparent, an efficient and robust closed attribute tree mining algorithm," *IEEE trans. on Knowledge and Data Engineering*, vol. 20, no. 3, pp. 300–320, 2008.

[7] A. da Jiménez, F. Berzal, and J.-C. Cubero, "Frequent tree pattern mining: A survey," *Intell. Data Anal.*, vol. 14, no. 6, pp. 603–622, Nov. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1890496.1890498

[8] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas, "Fast vertical mining of sequential patterns using co-occurrence information," in *Proc. of the 18th PAKDD Conference, Part I*, 2014, pp. 40–52.

[9] S. Hido and H. Kawano, "Amiot: induced ordered tree mining in tree-structured databases," in *Proc. of the 5th DM Conference*, 2005.

[10] H. Sleiman and R. Corchuelo, "A survey on region extractors from web documents," *IEEE trans. on Knowledge and Data Engineering*, vol. 25, no. 9, pp. 1960–1981, Sept 2013.

[11] C. Kohlschütter and W. Nejdl, "A densitometric approach to web page segmentation," in *Proc. of the 17th ACM CIKM Conference*, 2008, pp. 1173–1182.

[12] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne, "Finding high-quality content in social media," in *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM, 2008, pp. 183–194.

[13] J. Xu, T.-Y. Liu, M. Lu, H. Li, and W.-Y. Ma, "Directly optimizing evaluation measures in learning to rank," in *Proc. of the 31st SIGIR conference*, 2008, pp. 107–114.

[14] X. Song, J. Liu, Y. Cao, C.-Y. Lin, and H.-W. Hon, "Automatic extraction of web data records containing user-generated content," in *Proc. of the 19th CIKM Conference*, 2010, pp. 39–48.

[15] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser, "Extracting data records from the web using tag path clustering," in *Proc. of the 18th WWW Conference*, 2009, pp. 981–990.

[16] L. Bing, W. Lam, and Y. Gu, "Towards a unified solution: data record region detection and segmentation," in *Proc. of the 20th CIKM Conference*, 2011, pp. 1265–1274.

[17] J. Domenech, A. Pont, J. Sahuquillo, and G. J. A, "A user-focused evaluation of web prefetching algorithms," *Computer Communications*, vol. 10, no. 30, p. 22132224, 2007.

[18] K. Gyllstrom, C. Eickhoff, A. P. de Vries, and M.-F. Moens, "The downside of markup: examining the harmful effects of css and javascript on indexing today's web," in *Proc. of the 21st CIKM Conference*, 2012, pp. 1990–1994.

[19] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proc. of the 24th ICML Conference*, 2007, pp. 129–136.

[20] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li, "Listwise approach to learning to rank: theory and algorithm," in *Proc. of the 25th ICML Conference*, 2008, pp. 1192–1199.

[21] Y. Zhai and B. Liu, "Web data extraction based on partial tree alignment," in *Proc. of the 14th WWW Conference*, 2005, pp. 76–85.

[22] B. Liu, R. Grossman, and Y. Zhai, "Mining data records in web pages," in *Proc. of the 9th KDD Conference*, 2003, pp. 601–606.

[23] L. Bing, W. Lam, and Y. Gu, "Towards a unified solution: data record region detection and segmentation," in *Proc. of the 20th CIKM Conference*. New York, NY, USA: ACM, 2011, pp. 1265–1274. [Online]. Available: http://doi.acm.org/10.1145/2063576.2063761

[24] M. Yoshida, K. Matsumoto, K. Kita, and H. Nakagawa, "Unsupervised analysis of web page semantic structures by hierarchical bayesian modeling," in *Proc. of the 18th PAKDD 2014 Conference, Part I*, 2014, vol. 8444, pp. 572–583.

[25] D. Cai, S. Yu, J. Wen, and W. Ma, "Extracting content structure for web pages based on visual representation," in *Proc. of the 5th APWeb Conference*, 2003, pp. 406–417.