Extracting Records and Posts from Forum Pages with Limited Supervision

Luciano Barbosa and Guilherme Ferreira

IBM Research – Brazil
Av. Pasteur, 138, Rio de Janeiro, Brazil
{lucianoa,guiferre}@br.ibm.com

Abstract. Internet forums are rich sources of human-generated content. Many applications, such as opinion mining and question answering, can greatly benefit from mining and exploring such useful content. An important step towards making user content from forums more easily accessible is to extract it from forum pages. We propose REPEX (REcord and Post EXtractor), a two-step solution that uses limited supervision to achieve this goal. Given a forum page, REPEX first extracts data records that contain human-generated content and then, from these records, extracts their user content. The record extraction assumes that (1) a record is composed of an automatic-generated part, which we call record template, and a human-generated part; and (2) the structure of record templates are usually consistent across records. Based on those, the record extractor initially locates the subtree that contains all records in the forum page, using an information-theoretic measure, and then identifies the template of the records in this subtree, modelling this as an outlier detection problem. Finally, starting from the templates, REPEX determines the boundaries of the records. For the post extraction, REPEX applies an information extraction approach that performs this task by identifying the posts' string boundaries. We have performed experiments over more than 100 real forum websites, and the results show that REPEX is highly effective, obtaining high values of precision and recall for both tasks.

Keywords: Forum, Record extraction, Post extraction, Data mining.

1 Introduction

An internet forum is described on Wikipedia as an "online discussion site where people can hold conversations in the form of posted messages". These conversations address many different subjects and topics (e.g. games, movies, travel, computers, health etc), which makes their content very diverse. Web forums are also very popular nowadays. To give some numbers, as of March 2015, a big forum website – ConceptArt.org[1] – had more than 375 thousand users and more

[1] http://www.conceptart.org/

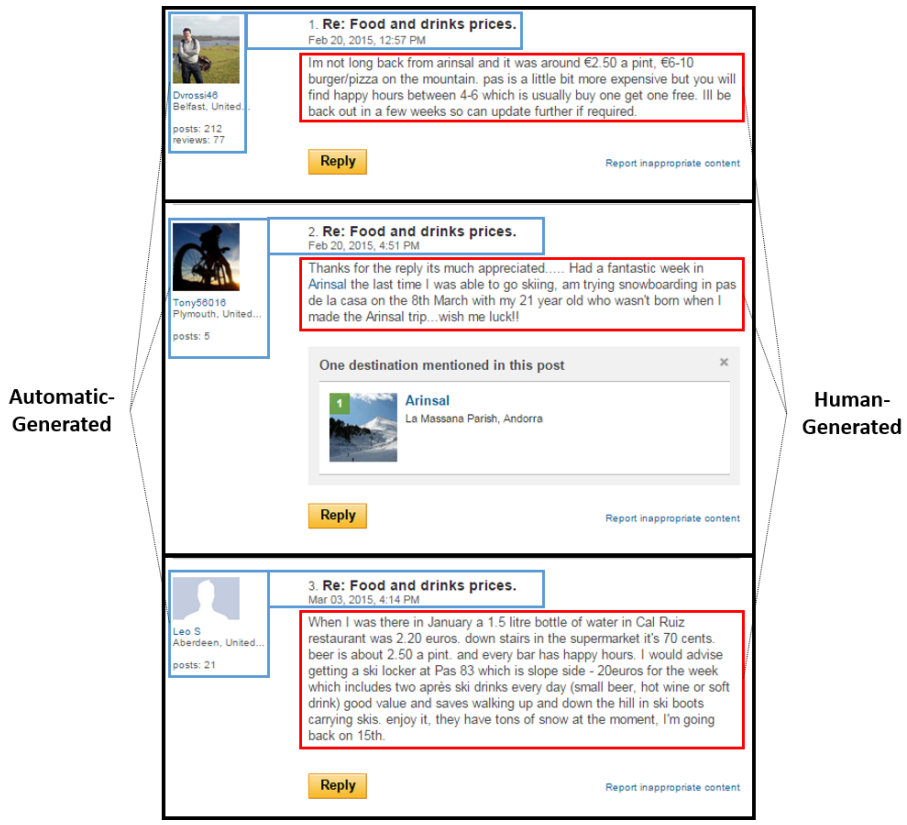

Fig. 1. Example of 3 records in a thread page.

than 8 billion posts, another forum – Gaia Online[2] – had about 26 million users and 2 billion messages (source: The Biggest Boards[3]). This huge amount of diverse human-generated content is very helpful for a variety of applications such as opinion mining [11], question answering [16] and forum search [12, 4].

To take advantage of such rich content, methods to collect and process forum data have been previously introduced [5, 15, 13, 1]. In this paper, we focus on the particular problem of extracting human-generated content from conversational pages of forums, also known as thread pages. Thread pages are composed of data records composed by: a human-generated part (the user post); and the automatic-generated (or template) part, that contains information such as date/time of the post, the user who posted it and the title of the posts. Figure 1 presents an example of records and posts in a thread page.

[2] http://www.gaiaonline.com/
[3] http://www.thebiggestboards.com/

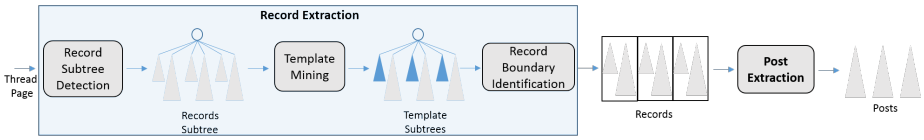

Fig. 2. Overview of REPEX's pipeline: given a thread page, Record Subtree Detection locates the subtree of records; Template Mining identifies the record template subtrees (in blue); and Record Boundary Identification determines the boundaries of each record; finally, from the records, Post Extraction extracts the posts.

Many previous approaches have been proposed to deal with the problem of web data extraction [2, 8, 13, 6, 20]. Our work is more related to [8, 13], since we are interested in building a solution that is not specific for a particular layout template (template-independent). We also want to perform this task with limited supervision, i.e., without any training data, to avoid having to label a large amount of data, which is a laborious and time-consuming task. The main challenge of building such solution is that the structure of thread pages vary significantly across forum sites. To deal with all this variability, we propose a two-step solution (see Figure 2). Given a thread page, our method first extracts data records that contain the user posts (Record Extraction), and then the posts within these records (Post Extraction).

To perform record extraction, we exploit common features present in the DOM tree of thread pages. More specifically, we make three assumptions: (1) records are under the same parent node in the HTML DOM tree of thread pages; (2) records contain an automatic-generated (or template) part, whose HTML structure is consistent across records in the same thread page; and (3) record templates contain common types across sites. Based on those assumptions, we propose a data mining algorithm that uses limit supervision (no training) to extract data records from thread pages: REPEX (REcord and Post EXtractor). First, given a thread page, REPEX scans its DOM tree to identify the candidate record templates by detecting nodes containing the following types: date and time of the post, the post user and the post title. We implement simple heuristic-based type detectors, which combined, provide a robust method for detecting the record template. Next, REPEX locates the subtree S where all records are located (Record Subtree Detection). For that, it assumes that S is balanced with respect to the detected type nodes, and uses an information-theoretic measure to calculate tree balance. Since not all candidate record templates within S are in fact record templates, next, it identifies them using an outlier detection strategy: candidates with very different tree structures are considered outliers and then removed (Template Mining). Finally, REPEX identifies the boundaries of the records in S, using a simple heuristic (Record Boundary Identification). Regarding post extraction, we assume that the string boundaries of the posts are similar across records and, based on that, we implement an information extraction approach to extract them.

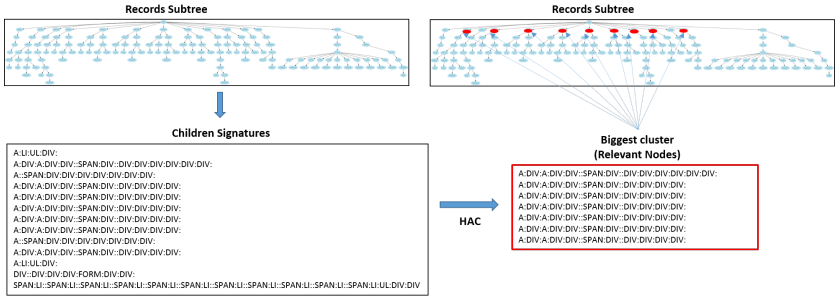

Fig. 3. Example illustrating how Template Mining selects template subtrees in the records subtree.

The remainder of the paper is organized as follows. Section 2 describes in details the record extraction, and Section 3 describes the post extraction. In Section 4, we detail the experimental evaluation. In Section 5 we present an overview of existing related work in the area of web data extraction, and finally in Section 6, we conclude the paper.

2 Record Extraction

The goal of the Record Extraction is to extract records from forum thread pages. The Record Extraction is composed of 3 sub-tasks: Record Subtree Detection, Template Mining and Record Boundary Identification. Record Subtree Detection locates the subtree in the DOM tree where all records are located. Within this subtree, Template Mining identifies the record templates and, based on the templates, Record Boundary Identification determines the boundaries of the records. Figure 2 gives an overview of this process. In this section, we first describe the type detectors used to identify the record templates and, subsequently, explain each one of the components of the record extraction.

2.1 Type Detectors

Our first assumption regarding the problem of record extraction is that a record contains a template-generated part, composed of basic types: the date and time that the record was posted, its title and the user who posted it. Based on that, we implemented 3 type detectors to identify them in a thread page: date-time, user and record title. The date-time detector was built from regular expressions. For that, we started from date and time examples of regular expressions available on specialized websites[4]. Then, we improved the quality of these expressions using a validation set, described in Section 4. The user detector uses simple heuristics

[4] http://www.regxlib.com/
http://www.regular-expressions.info/

Algorithm 1 Record Subtree Detection

```
1: procedure TreeScan(node, D_1...D_n)
2:     for each d in D_1...D_n do                          ▷ Type detection loop
3:         if TypeDetection(d, node) = TRUE then
4:             IncrementTypesCount(node)
5:         end if
6:     end for
7:     for each child node c of node do
8:         TreeScan(c)
9:     end for
10:    if Balance(node) > τ then                           ▷ Balance verification
11:        {t_1, ..., t_n} = TemplateMining(node)
12:        {r_1, ..., r_n} = ExtractRecords({t_1, ..., t_n})
13:    end if
14:    Return {r_1, ..., r_n}                              ▷ Extracted records
15: end procedure
```

Algorithm 2 Template Mining

```
1: procedure TemplateMining(node)
2:     {tc_1, ..., tc_z} = Children(node)                  ▷ Template children
3:     {s_1, ..., s_z} = Signatures({tc_1, ..., tc_z})     ▷ Signatures of the children
4:     {c_1, ..., c_l} = HAC({s_1, ..., s_z}, γ)           ▷ Clustering
5:     {t_1, ..., t_n} = max({c_1, ..., c_l})              ▷ Max size cluster
6:     if Size(maxCluster) > 2 then
7:         res = {t_1, ..., t_n}
8:     end if
9:     Return res
10: end procedure
```

to detect user information. It checks for URLs with words such as "'member"', "profile" and "user". The title detector assumes the record title is similar to the title of the thread page. To measure that, we calculate the Jaccard similarity between the title of the thread page and a given text. We consider a similarity of 0.3 as a match. The text used as input to the detectors is segmented based on the DOM tree structure: all the text within a leaf text node is considered as a single sentence.

The great advantage of using a set of detectors, instead of a single one as in [13], is that individual detectors can complement each other, and consequently produce better results. For instance, the user detector might work in sites in which the date-time detector might not. Another advantage is that building a strong detector is a laborious task. Thus, instead of having a single strong detector, one can build weak detectors, which need less effort to be implemented. Our experimental evaluation confirms all these observations.

2.2 Record Subtree Detection

The first step of Record Extraction is Record Subtree Detection. Given the thread page's DOM tree T, it identifies the subtree T' of T that contains all the records. To achieve this goal, we assume that the nodes that contain the template data types (date-time, user and title) are evenly distributed throughout the child subtrees of T'. Concretely, the algorithm works as follows (see Algorithm 1). First, given T, the algorithm performs a complete scan of T, labelling nodes that match the three basic types (lines 2-8). When a type detector matches a node (line 3), the types' counter in that node is incremented (line 4). Next, based on these counts, for each subtree T' in T, it measures how balanced the child subtrees CS of T' are with respect to the detected data type nodes. Only T's with a balance value higher than a threshold are considered candidate subtrees for the next steps (lines 10-12). The tree balance is measured using an information-theoretic approach. More formally, consider p the probability of a child subtree c of T' having detected data type nodes. We calculate p of c by dividing the number of detected nodes in c over the total number of detected nodes in T'. If T' is balanced, the entropy of T' would be high, since p for all children would have a similar value. To have a value between 0 and 1, we define $Balance$, which is the normalized entropy of T':

$$Balance(T') = -\frac{\sum_{c \in CS} p_c log(p_c)}{log(|CS|)} \quad (1)$$

The Record Subtree Detection works as a lightweight filter and, as the experimental results in Section 4 suggest, can considerably prune the search space for the next step in the pipeline, which is more expensive.

2.3 Template Mining

The goal of Template Mining is to identify the template part of the records. For that, we assume that the data types are more concentrated in nodes belonging to the template part of records than in other parts. Another assumption, similar to [13], is that the tree structure of the templates of the records is similar to each other. Based on these observations, we model this task as an outlier detection problem, in which detected nodes outside the template part of records are considered outliers. The algorithm works as follows (see Algorithm 2 and Figure 3 for a concrete example). Initially, given the subtree T', identified in Record Subtree Detection, the algorithm obtains the candidate templates CT, i.e., the children of T' that contains detected nodes (line 2). For each child c of CT, it generates a signature composed of the HTML tags of the detected nodes of c in the depth-first search order (line 3). This signature represents a flat representation of the tree structure of c with respect to its detected nodes. Next, these signatures are provided as input to the Hierarchical Aglomerative Clustering (HAC) [14] (line 4). HAC starts with $|CT|$ clusters (a single cluster corresponds to a child signature), where $|CT|$ is the number of elements of CT. The two closest clusters are merged, resulting in $|CC| - 1$ clusters. Next, the

Algorithm 3 Record Boundary Identification

```
1: procedure ExtractRecords({t_1, ..., t_n})
2:     sizeFreqs = [0, ..., 0]                          ▷ Record size calculation
3:     for each t_i in {t_1, ..., t_n} do
4:         sizeFreqs[t_i − t_{i−1}] = sizeFreqs[t_i − t_{i−1}] + 1
5:     end for
6:     recordSize = max(sizeFreqs)
7:     leftShift = 0                                    ▷ Left shift
8:     for j = t_1,k = t_2 to j ≥ 0 and k ≥ 0 do
9:         if EqualChildSign(j, k) = FALSE then
10:            break
11:        end if
12:        leftShift = leftShift + 1
13:        j = j − 1
14:        k = k − 1
15:    end for
16:    {r_1, ..., r_n} = ∅                              ▷ Segmentation of records
17:    for each t_i in {t_1, ..., t_n} do
18:        start = t_i − leftShift
19:        end = start + recordSize
20:        r_i = t_start, ..., t_end
21:    end for
22:    Return {r_1, ..., r_n}
23: end procedure
```

two closest of the $|CC| - 1$ clusters are merged, and then the process continues until a stop condition. The output of this process is a set of clusters. The algorithm considers that the record templates are in the cluster with the highest number of elements (line 5), and the remaining clusters are discarded. The subtrees belonging to this cluster are returned, if the cluster has more than 2 elements (lines 6-8). We adopted as stop condition a similarity threshold, defined experimentally. We use as similarity measure the levenshtein distance [10].

2.4 Record Boundary

Template Mining selects the template subtrees $\{t_1, ..., t_n\}$ in T'. These subtrees, however, do not necessarily contain the whole record. There might be cases, for instance, in which the human-generated content of a record is in a separated subtree of T'. In other cases, a record might be composed of multiple subtrees of T'. The task is, therefore, to define how we segment the children of T' in order to extract the records (see Algorithm 3). First, the algorithm determines the record size, i.e., how many consecutive child subtrees of T' compose a record. It does so by calculating the distance (i.e., how many subtrees are) between each pair of consecutive template subtrees (lines 2-5). It considers the distance with the highest frequency as the record size (line 6). Then, it defines in which position to the left of the template subtree the records start (lines 7-15). For that, it goes

Algorithm 4 Post Extraction

```
1: procedure ExtractPosts({r_1, ..., r_n})
2:     for each r_i in {r_1, ..., r_n} do                          ▷ Pre-processing
3:         SegmentRecord(r_i)
4:         DetectPostText(r_i)
5:     end for
6:     p, r = GetMaxRecordAndIndex({r_1, ..., r_n})   ▷ Position p of record r chosen.
7:     {s_i, ..., s_n} = 0                                          ▷ Begin indexes
8:     for left = p − 1 to left ≥ 0 do
9:         text = record[left]
10:        indexes = MatchPositions({r_1, ..., r_n}, text)
11:        if Size(indexes) = n then
12:            {s_i, ..., s_n} = indexes
13:            break
14:        else
15:            {s_i, ..., s_n} = 0
16:            left = left − 1
17:        end if
18:    end for
19:    {e_i, ..., e_n} = 0                                            ▷ End indexes
20:    for right = p + 1 to right ≥ 0 do
21:        text = record[right]
22:        indexes = MatchPositions({r_1, ..., r_n}, text)
23:        if Size(indexes) = n then
24:            {e_i, ..., e_n} = indexes
25:            break
26:        else
27:            {e_i, ..., e_n} = 0
28:            right = right + 1
29:        end if
30:    end for
31:    {p_1, ..., p_n} = ∅                                  ▷ Segmentation of posts
32:    for i = 1 to n do
33:        p_i = r_start[i], ..., t_end[i]
34:    end for
35:    Return {p_1, ..., p_n}
36: end procedure
```

backward from the first two template subtrees (t_1 and t_2) until it finds subtrees of T' with different child signatures (line 9). We define a child signature as the string composed of the subtree HTML tag concatenated with the tags of its children. Finally, the records are extracted from T' for each template subtree t_i (lines 16-21).

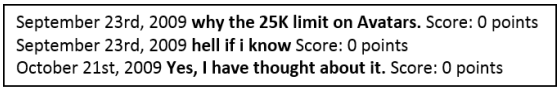

Fig. 4. Examples of texts from 3 records. Posts are highlighted.

3 Post Extraction

The Post Extraction is the final step of REPEX (see Figure 2). It extracts the human-generated content from the data records. Instead of only relying on the DOM tree structure to perform this task, as we did for record extraction, we handle this task as an unstructured information extraction problem. For that, we look at regularities in the text resulting from the records. Figure 4 shows examples of texts extracted from records. In these records, posts (in bold) are located between a date string (e.g. "September 23rd, 2009") and the string "'Score"'. This illustrates the main assumption of this algorithm: posts are delimited between common types/strings across records. The goal of Post Extraction is then to identify these delimiters, and then extract all text between them. Concretely, the algorithm works as follows (see Algorithm 4). First, it segments the record sentences, using their structure on the DOM tree (line 3). All the text in the same leaf node is considered a single sentence. Next, it runs a post-text detector over the sentences (line 4). Similar to the other detectors presented in this paper, the post-text detector uses simple rules to perform the detection. For instance, it looks for characters such as "." or "?" at the end of the phrase along with personal pronouns as "I" or "you" or "it". Here we assume that the text in posts have a good chance of having personal references. From all the records with detected texts, the algorithm selects the one that it has a high confidence of having in fact a post text (lines 6): the record r with the largest detected text in phrase position p in r. From p, the algorithm goes backwards until it finds a type/string in r that matches in all others records (lines 7-18). The positions $\{s_1, ..., s_n\}$ of these matches in the records represent where the posts should start. Conversely, the algorithm does the same procedure going forward from p (lines 19-30). The positions $\{e_1, ..., e_n\}$ of these matches in the records represent where the posts should end. To perform this match, it verifies whether the strings are from the same data type (date-time, user and title) or if they share some prefix of size greater than 1. Finally, it extracts the posts using $\{s_1, ..., s_n\}$ and $\{e_1, ..., e_n\}$ as delimiters (lines 31-34).

4 Experimental Evaluation

In this section, we present the experimental evaluation of our proposed approaches.

4.1 Experimental Setup

Data. To perform the evaluation, we collected thread pages from 118 forum sites. These websites are in a variety of different topics: games, cancer, psychology etc. In addition, similar to [12], we also tried to select as many forums as possible that use different softwares to publish their content. Out of the 118 sites, 72 were used in the validation set and 46 in the test set. For each one of the websites, we collected at most 5 thread pages, resulting in a set of 282 pages in the validation set and 200 in the test set. Then, we manually extracted the text in the records and posts from these pages, resulting in a total of 2,449 records and the same number of posts. The performance of an approach is measured by comparing its output with the gold data. Since there might be small differences between the way records and posts are extracted, we consider a match when the cosine similarity between the approach's record and the gold data's record is higher than 0.6 for records and 0.3 for posts.

Record Extraction Approaches. We compare our record extraction approach with the state-of-art for this task: MiBAT [13], which has been obtained superior performance compared to previous approaches. MiBAT uses a date-time detector to identify the template part of the records, which they call anchor trees. Then it aligns anchor trees using a tree matching algorithm [18]. The matched anchor trees compose the templates of the records. For this matching, the authors proposed similarity measures. We used Pivot and Siblings (PS) similarity, since it showed the best results in their experiments. A similarity higher than a given threshold is considered a match. We used the validation set to tune this parameter. For further details, we refer the reader to [13]. We also used the validation set to tune two parameters of our approach: the minimum entropy of a parent node being considered relevant, and the similarity threshold in the HAC algorithm's stop condition.

Post Extraction Approaches. In addition to the post extraction approach proposed in this paper, which will we call String-based Extraction for the remaining of this section, we implemented two other strategies:

- Text Detection: this approach scans the records, and only considers as posts the text detected within the records by the Text Detector.
- Tree-based extraction: this algorithm works as follows. Given the subtree that contains all the records T', first it uses the Text Detector to identify text nodes in T'. For the child subtrees CS of T' that contain text nodes, it identifies the largest common subtree LCS of all CS. Since we assume the post part of the record subtree might not have much regularity, for each record, the algorithm considers the post part the tree structure of the record that does not belong to the LCS. This method has not been proposed previously in the literature. We implemented it to have a reasonable baseline for post extraction.

	Rec	**Prec**	**F-Measure**	**Prop. of Pages**
RecExt	0.94	0.92	0.93	0.97
MiBAT	0.51	0.94	0.66	0.53

Table 1. Recall, precision, F-Measure and proportion of pages with at least 1 record extracted by each approach.

	Rec	**Prec**	**F-Measure**
User,Date-Time	0.86	0.92	0.89
Date-Time,Title	0.82	0.93	0.87
User,Title	0.76	0.96	0.85
User	0.67	0.96	0.79
Date-Time	0.68	0.93	0.79
Title	0.32	0.99	0.48

Table 2. Results of our approach using different combinations of type detectors.

4.2 Record Extraction Results

For each approach, we measured precision, recall and F-Measure over the records in the test set. We also calculated the proportion of pages that had at least 1 record extracted by each approach. Table 1 presents the results. Our approach obtained high values of recall (0.94), precision (0.92), F-measure (0.93), and also extracted records from the vast majority of the pages (0.94). The numbers also show that our approach outperforms the baseline in all measures.

We investigated possible causes for this difference in performance. For that, we calculated the performance of MiBAT over only the 53% of the test set that it was able to extract records. As expected, its results are much better: recall = 0.92, precision = 0.97 and F-Measure = 0.95. For comparison, we also ran our approach over the same 53% set. It obtained recall = 0.96, precision = 0.95 and F-Measure = 0.96. Our approach obtained higher recall (0.96 vs 0.92) but lower precision (0.95 vs 0.97). Overall, our approach obtained a slightly better F-Measure (0.96 vs 0.95).

We also evaluated the contribution of each detector for the final result. Table 2 presents the recall, precision and F-Measure for all the possible combinations of detectors. The combination of user and date-time detectors obtained the best results as well as these two detectors considered individually. Although the title detector individually obtained a poor result in terms of recall, combining it with the other detectors, it boosted the overall performance of our approach. These numbers clearly show that a combination of "weak" detectors that complement each other, i.e., covering different sets of pages, leads to an effective extractor.

Since MiBAT only uses a single date-time detector, we can compare its performance in Table 1 with our approach using only this detector (Table 2) over the entire test set. Our approach got a much higher recall than MiBAT (0.68 vs 0.51)

	Rec	**Prec**	**F-Measure**
String-based Extraction	0.86	0.93	0.89
Tree-based Extraction	0.82	0.92	0.87
Text Detection	0.57	0.56	0.56

Table 3. Results of post extraction.

and a slightly smaller precision (0.93 vs 0.94), as a result a higher F-Measure (0.79 vs 0.66). The main reason for this advantage in recall is that our approach, with only a single date-time detector, was able to detect date-time nodes from a higher proportion of pages: 0.68 vs 0.51. From this, we can conclude that MiBAT was not able to extract records even in pages that the date-time detector worked.

In terms of performance, a major difference between our approach and MiBAT is that the record region identification considerably prunes the search space, i.e., only a small fraction of the subtrees in a page is considered for the remaining steps in the pipeline. MiBAT, on the other hand, analyzes all the subtrees in the page. To give some numbers, if one considers the total number of subtrees in all pages in the test set (more than 500K subtrees), the Record Subtree Detection only considered 1.1% of them as relevant, and then only this small fraction was considered to the next steps.

4.3 Post Extraction Results

The results of the post extraction approaches are presented in Table 3. The String-based approach obtained the highest values of recall (0.86), precision (0.93) and F-Measure (0.89), followed by the Tree-based approach. The numbers show that our approach of post extraction is in fact effective for this task. The lowest result was obtained by the approach that only uses the text detector to extract the posts. The main reason for this poor performance is that a reasonable portion of the text in posts are not detected by the Text Detector (low recall), and also much of the text detected by the Text Detector does not belong to the posts (low precision). We can conclude from this that using the text detection itself is not enough for this task, but it is very useful when used with our proposed strategy. Regarding the recall of all approaches, an important observation is that the post extraction is performed after the record extraction. As a result, the upper bound of recall is the one obtained by our record extraction technique: 0.94. The precision of the record extraction also has influence over the precision results for post extraction.

5 Related Work

There has been many different approaches in the literature regarding extracting data records from the Web [3].

Liu et al. [8] proposed a fully-automated algorithm based on similar sub-tree mining on the DOM tree of the Web page, called MDR. Although MDR obtains good results, it suffers from a few limitations, such as only being able to extract a consecutive record list and a poor similarity measure. Li et al. [7] propose an approach using HTML tag paths and visual text features for record extraction. They also assume records are continuous. In contrast to them, REPEX does not make any assumption that records are consecutive. Zhang et al. [19] present a novel algorithm to text extraction. Their approach differs from ours as it relies heavily on visual properties of the webpage, which are harder to extract. Yang et al. [17] use both page-level and site-level knowledge in Markov logic networks to extract structured data from data records, such as post title, post author and post time. As opposed to our approach, they heavily rely on labeled data, since they are applying a statistical model.

Miao et al. [9] focused on extracting data records by capturing a list of objects using a comparison between a pair of tag path occurrence patterns to estimate how likely these two tag paths represent the same list of objects. Another proposed approach for the problem of record extraction – MiBAT [13] – uses domain constraints, such as post-date, to improve the extraction process and to design better similarity measures. They used only a post-date constraint for this task, implemented by a post-date detector. As opposed to them, we applied a broader set of type detectors. The advantage of doing this is that multiple detectors can complement each other and, as result, less effort needs to be done to build them.

6 Conclusions

In this paper, we present REPEX, a solution for extracting data records and user posts from forum pages that does not use any training data to perform this task. The data record extraction assumes that data records are within the same subtree in the page's DOM tree, and records have a template part that has a similar tree structure. To locate the data record subtree, it uses an information-theoretic approach. Next, within this subtree, it identifies the template part of the records using a clustering algorithm. Finally, it determines the boundaries of the records expanding from the templates. The extracted records are then passed to the post extraction, that uses an unstructured information extraction strategy to define the boundaries of posts, and extract them. Our solution has been shown to be very effective over real forum data and outperformed the baselines in different scenarios.

References

1. G. Cong, L. Wang, C.-Y. Lin, Y.-I. Song, and Y. Sun. Finding question-answer pairs from online forums. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 467–474. ACM, 2008.

2. D. W. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in web documents. In *ACM SIGMOD Record*, volume 28, pages 467–478. ACM, 1999.
3. E. Ferrara, P. De Meo, G. Fiumara, and R. Baumgartner. Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, 70:301–323, 2014.
4. G. Ganu and A. Marian. One size does not fit all: Multi-granularity search of web forums. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 9–18. ACM, 2013.
5. J. Jiang, X. Song, N. Yu, and C.-Y. Lin. Focus: learning to crawl web forums. *Knowledge and Data Engineering, IEEE Transactions on*, 25(6):1293–1306, 2013.
6. K. Lerman, C. Knoblock, and S. Minton. Automatic data extraction from lists and tables in web sources. In *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, volume 98, 2001.
7. S. Li, L. Tang, J. Hu, and Z. Chen. Automatic data extraction from web discussion forums. In *Frontier of Computer Science and Technology, 2009. FCST'09. Fourth International Conference on*, pages 219–225. IEEE, 2009.
8. B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–606. ACM, 2003.
9. G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. Extracting data records from the web using tag path clustering. In *Proceedings of the 18th international conference on World wide web*, pages 981–990. ACM, 2009.
10. G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19–27, 2001.
11. B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.
12. J. Seo, W. B. Croft, and D. A. Smith. Online community search using thread structure. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 1907–1910. ACM, 2009.
13. X. Song, J. Liu, Y. Cao, C.-Y. Lin, and H.-W. Hon. Automatic extraction of web data records containing user-generated content. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 39–48. ACM, 2010.
14. P.-N. Tan, M. Steinbach, V. Kumar, et al. *Introduction to data mining*, volume 1. Pearson Addison Wesley Boston, 2006.
15. H. Wang, C. Wang, C. Zhai, and J. Han. Learning online discussion structures by conditional random fields. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 435–444. ACM, 2011.
16. B. Webber and N. Webb. Question answering. *The handbook of computational linguistics and natural language processing*, pages 630–654, 2010.
17. J.-M. Yang, R. Cai, Y. Wang, J. Zhu, L. Zhang, and W.-Y. Ma. Incorporating site-level knowledge to extract structured data from web forums. In *Proceedings of the 18th international conference on World wide web*, pages 181–190. ACM, 2009.
18. W. Yang. Identifying syntactic differences between two programs. *Software: Practice and Experience*, 21(7):739–755, 1991.
19. Q. Zhang, Y. Shi, X. Huang, and L. Wu. Template-independent wrapper for web forums. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 794–795. ACM, 2009.

20. J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. Simultaneous record detection and attribute labeling in web data extraction. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 494–503. ACM, 2006.