# Automating Blog Crawling Using Pattern Recognition

Anal Kanti Roy[1], Nitin Agarwal[2]
Department of Information Science
University of Arkansas at Little Rock
Little Rock, Arkansas, USA
e-mail: [1]axroy@ualr.edu, [2]nxagarwal@ualr.edu

***Abstract*—Social media plays an important role in the propagation and dissemination of ideas and thoughts leading to the formation of diverse online communities. Compared to a myriad of other social media sites and applications, blogs provide a convenient platform for users to post detailed information, engage in active discussions and share the content on other social media sites, such as Facebook and Twitter. Thus, the blogosphere has been an enormous and ever-growing part of the open-source intelligence. In order to track and monitor online social behavior particularly from blogs, the first challenging part is to mine the vast pool of unstructured data. Several approaches have been developed to extract blog data using focused crawling, which requires a lot of time, effort and manual intervention. To scale up this process and cope with the continuously changing blog structure, we propose a sophisticated, advanced, generic, and scalable automated blog-crawler, with ability to identify different patterns in the Hypertext Markup Language (HTML) structure of the blog pages and extract data, such as title, author, date, content, tags, etc. from different blog posts. Using the crawler, we have crawled 530 blog sites with 894,856 blog posts so far.**

***Keywords- blog crawling; generic crawler; blogs; blog posts; metadata; title; author; date; content; patterns; html.***

## I. INTRODUCTION

Recent years have witnessed an explosive growth of social media driven communications. Due to the pervasive nature of social media platforms, people have become more engaged in their ways of expressing thoughts. Usage of social media is no longer limited to just networking, advertising or self-expression. Sometimes, the negative side of the social platforms are encouraging people to utilize these online platforms for malicious purposes, such as spreading rumors, propaganda, polarization, extremism and radicalization.

The blogosphere (the clustered network of blogs and comments in existence on the internet and their links to other social media platforms) [1] is considered a highly dynamic subset of the social media. Starting from 1994, from early blogging platforms like LiveJournal, TypePad, and Blogger, the blogosphere has grown considerably with the addition of services like Tumblr, WordPress, Medium, Squarespace, etc. more than 500 million are recognized as blogs. Their authors account for over 2 million blog posts daily. Tumblr, possibly the biggest blogging platform, reports that it hosts over 440 million blogs. The most popular Content Management System (CMS), WordPress, adds about 60 million more [2]. Apart from the massiveness, this domain also distinguishes itself from the rest of social media platforms due to several features, such as more space for building discourse, more involvement in the conversation, and ability to spread into other platforms through shares. Analyzing blogs would therefore provide insights into our cyber behaviors, whether it is to monitor cyber campaigns, identify powerful actors and groups, study propaganda dissemination, and trace cyber threats [3][4]. However, collecting blog data is imperative for conducting any sort of computational analysis. This data collection process is quite challenging due to several reasons. First, the majority of information that can be crawled from blogs is unstructured and noisy, which makes it difficult to predict and model the crawling process. Second, the problem of automated crawling is exacerbated by the enormous growth rate of blog data.

To address the challenges mentioned above, we developed a generic crawler that is able to automatically parse the blog information for any blog site, categorize different data types based on their respective patterns, clean the data through data munging module and finally storing them in organized format in a Database Management System (DBMS). Although there is room for improvement for the automated crawler, it surely enhances the effectiveness and scalability of blog data collection. The rest of this paper is organized as follows. Section II describes similar blog crawling approaches taken by others so far. Section III explains separately how each type of blog data, such as title, author, date and content are extracted using our rule-based approach. The conclusion and acknowledgement close the article.

## II. LITERATURE REVIEW

This literature review summarizes few blog-specific crawling methods available on the web and their drawbacks.

### *A. Web Content Extractor (WCE)*

WCE is a crawling tool designed to extract web's data in general. For the extraction of data from the blogs, the patterns of the HTML pages and the patterns of the different data entities (title, author, date, etc.) has to be manually fed into the WCE before crawling any blog(s). Although WCE is quite accurate and can be operated without any prior knowledge of programming, but it has to be distinctly set up for every individual blog-site, which certainly is not a scalable solution when targeting multiple blogs. WCE has the feature of exporting the data into different formats .xml, .csv, .txt etc.) [5].

 ISBN: 978-1-61208-757-3 

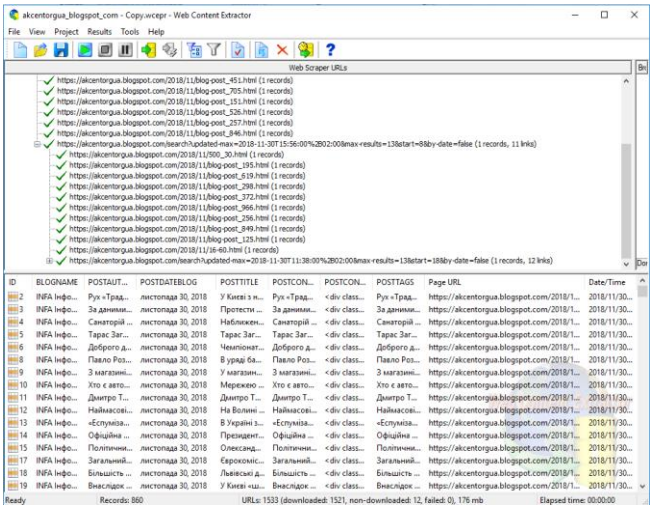

Figure 1. Extraction of blog elements by Web Content Extractor [5]

However, it does not have a synchronization ability with the database management systems. Therefore, a user has to convert the exported files manually into a compatible format that suits the database. It is also a multi-threaded web crawler that supports the data extraction from up to 20 threads (webs) simultaneously (Figure. 1 Extraction of blog elements by Web Content Extractor). However, the speed of the crawling process is normal as general crawler and sometimes slow while executing huge scripts. Moreover, this tool is completely focused and bound to the limited tasks with less room for customizations [5]. Therefore, WCE is quite not suitable to collect enormous amounts of data from a huge number of blogs.

### *B. Mapping the blogosphere towards a universal and scalable Blog-Crawler*

The crawler in this method identifies the posts by crawling only feeds, RSS and atom from a host identified as a blog [6]. The crawler identifies the blog by downloading the crawled webpage and then parse it to check for the common standard patterns of the diverse blog systems. For instance, a pattern could be a match, if the generator tag of a web page contains < meta content = ‘‘blogger’’ name = ‘‘generator’’ />. Then crawler downloads the first alternate link rel = ‘‘alternate’’ with the type of an eXtensible Markup Language (XML) page recognized by type = ‘‘application / RSS + xml” or type=”application/atom+xml” and stores the referenced feed as the main feed of a blog, which should contain all posts. In some cases, the crawler may end up crawling the feeds, which contains only a minimized version of all posts, which are displayed as summaries on the home page of a blog(s). These instances occur because few web pages are limited to only displaying the latest posts. Consequently, the crawler has a drawback of crawling only the subsets of the displayed posts. The uncovered posts create a historical gap, as the crawler is unable to dive deep into the archives [6].

### *C. A New Algorithm of Blog-oriented Crawler*

This work aims at downloading blog pages in some portal websites and the crawler views the blog as the special "topic" [7]. The concept of this method is identical to the topical crawling. Generally, topic describes the contents of a web page and is defined by the users before being fed as an input into the crawler. Similarly, for a blog as a topic, it refers to the types of blog pages and describes the structural patterns of those pages. It also can differentiate whether the fetched pages are relevant to predefined topic or not, and then the pages are downloaded if they are relevant, otherwise dismissed. The strategy of the topical crawler is to check for the contents of pages or structure of linkages. It prunes the URLs when crawling to some extends and orders the handled URLs. This crawler is neither a depth-first search nor a breadth-first search, but the best-first search. That means the crawler downloads the most relevant page, which the URL points to, in a current situation. This method also have a mechanism to extract blog linkages. However, it has some drawbacks in terms of precision and efficiency because of its generality and no specialty in nature [7].

### *D. BlogPulse*

BlogPulse [8][9][10] devised by Intelliseek, was designed to find trends and patterns across selected 22,000 weblogs. It provided a list of key phrases, key persons and key paragraphs every day. It also had several analytical features that included gathering the insights by tracing the daily activities from the blogs, tracking the information by studying how the topic of information was disseminated through the blog posts. It profiled the top blogs, which provided detailed information about how the blog influences others and their activities. It extracted the list of 22,000+ weblogs from the BlogStreet directory. These blogs were crawled every day and were checked for possible duplication. The major limitation with BlogPulse is the restriction on the number of blogs available for crawling. Another challenge with the approach was that it was not user-friendly. Users were not allowed to monitor some set of blog sites and gather the trend information. Hence, the tool was a one-way reporting tool. It also does not fetch the comments from the blog posts, while crawling and mining the data. This tool was deprecated in 2012 [9].

Therefore, blog(s) oriented data retrieval systems are still in their premature stages. Blog-specific search engines only index the feeds, which usually are readable XML versions of the blogs. Most of them only provides a summary of the whole context obtained from the blog entries [11]. Some of them either focus on a tiny subset of blog(s) data with similar patterns [10], which in-turn consumes too much time and manual effort to crawl. The generic crawler we proposed allows us to crawl all the blog pages active in any blog irrespective of its different web structures saving significant amount of time and manual intervention which makes it different from other approaches.

 ISBN: 978-1-61208-757-3 

## III. METHODOLOGY

Since, the websites do not follow a unified standard that would allow the computer to semantically understand what every bit of HTML is trying to say, there is no common approach for grabbing patterns for every metadata. The ideal solution would be to build a robust and scalable tool that will be able to extract data irrespective of HTML content and structure. That is why; training the crawler for different HTML patterns of blogs, along with machine learning techniques should be an appropriate approach to apply. It will let the computer to understand what are on a HTML page exactly in the same way a human would.

First, we need an algorithm to recognize any blog pages. Among several papers [4][7][12] published, "Blog identification and splog detection" [12] suggests handful algorithms to identify blog pages by certain features like blog mark in URL, RSS tag, ordered dates in log etc.

### A. Link extraction

After the recognition of blog pages, the next step is the extraction of all the relevant links (URLs) and identifying the patterns for required data. Here is the step-by-step approach for link extraction.

- Start crawling the URLs in every page of a blog site using Scrapy [13] that crawls in a DFO order.
- Filter out external URLs using Scrapy's rule-based approach [13], which only look for the domain name of the blog we are crawling (1).
  rules = (Rule (LinkExtractor (allow = [keyword], callback = 'parse_item', follow = True),) (1)
- Out of the internal URLs, it tries to eliminate the ones with a set of stop-words. Here goes the list:
  stopwords = ['facebook.com', 'google.com', 'twitter.com', 'youtube.com', 'pinterest.com', 'instagram.com', '/page/', '/search', '/author/', '/tag/', '/category/', '/about', '/comments/', '/contact/', '?', '=']

### B. Extraction of Blog Content

Our method is to follow rule-based approach to recognize a specific information in a blog post. For example, 'title' of a blog post has a set of features or parameters that may classify it from any other piece of information. These features can be termed as classifiers. In order to filter out the post title, every chunk of data in the blog page has to go through the validation of these classifiers. Each classifier will provide them a score on a scale of 0 to 1 based on how they satisfy the conditions. The sum of scores from all the classifiers is the total score of each chunk of data. The chunk with highest score is considered as the post-title only if it surpasses a certain threshold. Since the importance or occurrence of all the classifiers is not the same, we multiply individual classifier scores by different weights before adding them. The weights can be determined by Naive Bayes Classification algorithm. We generated generic module for different data types, such as title, author, date and content. Here goes the explanation of step-by-step approach for each module.

### C. Extraction of post title

Collect every chunk of text from the blog page by filtering out the empty nodes and store them in a list. Then perform required string manipulation in the chunks like stripping, joining and getting rid of unnecessary scripts. In this case, the classifier is set with the following patterns:

- The post titles are mostly likely to be surrounded by h1 or h2 tags.
- In blog site analysis, it can be observed that a post titles are not longer than 200 characters. Hence, this classifier filters out the descriptive chunks from the pool.
- The titles commonly appear at the beginning of the page. Therefore, the nodes with less depth in the HTML tree gets high priority.
- The post titles have a great chance to completely or partially match with the text between tags. The match percentage between these two texts could possibly lead to a decision. To match the similarity between the strings, we choose Cosine Similarity measurements here.
- In most of the cases, the URLs of the blog posts contain words quite similar to the post title delimited by '-' or '/'. Therefore, if we match the words of each chunk with the split words from the URL, the result significantly can contribute to a decision making.
- The post titles are often surrounded by tags with a class or id name of the "title", or a name where title is a substring. These scenarios can be used to identify the title quite easily.
- Most of the cases, the URLs of the blog posts contain words quite similar to the post title delimited by '-' or '/'. Therefore, if we match the words of each chunk with the split words from the URL, the result significantly can contribute to a decision making.
- The post titles do not usually end with a full stop. This filter out the other chunks from consideration.
- These individual scores are multiplied by their respective weights and then added up to obtain the total score.
- Equation (2) below shows that the chunk with highest score is extracted as post title only if it exceeds a certain threshold (3).

  Target = max (sum (individual scores * weight)) (2)

  Post title = TRUE if (Target > Threshold) (3)

- Now, the question is how the weights and the threshold are calculated. In this regard, we incorporate a data analysis over our collected feed

from a sample set of blog pages. Then, we use Naive Bayes classification algorithm.

TABLE I. CLASSIFIER TABLE

| | Classifiers | True/False |
|---|---|---|
| ***Page 1*** | Between h1 tags | True |
| | Matches title tag text | True |
| | Class or id named title | True |
| ***Page 2*** | Between h1 tags | True |
| | Matches title tag text | True |
| | Class or id named title | False |
| ***Page 3*** | Between h1 tags | True |
| | Matches title tag text | False |
| | Class or id named title | False |

TABLE II. FREQUENCY TABLE

| Classifiers | True | False | Total |
|---|---|---|---|
| Between h1 tags | 3 | 0 | 3 |
| Matches title tag text | 2 | 1 | 3 |
| Class or id named title | 1 | 3 | 3 |

TABLE III. LIKELIHOOD TABLE

| Classifiers | True | Likelihood | Normalized-*3 (Weights) |
|---|---|---|---|
| Between h1 tags | 3 | 1.00 | 3.00 |
| Matches title tag text | 2 | 0.67 | 2.00 |
| Class or id named title | 1 | 0.33 | 1.00 |

a. Rule-based classifiers for blog pages [14]

Let us assume that data are collected according to the performance of three classifiers from three blog pages where post title is taken as an example. Tables I, II and III show how weight is calculated. Likelihood refers to the chances of occurrence of a pattern. The pattern with more weight is given more significance while calculating the total score. From the above analysis, we calculate the threshold from the minimum total score that qualified as a post title, if the classifiers guessed correctly [14].

### *D. Extraction of post author*

For any fetched URL, our method is to cross check with five predefined patterns, which identifies the author of a blog post. On passing the rules, each block of text gains a score just like post title extraction and then each score can be multiplied by their respective weights. Weights are determined by precision and recall of the training data. Finally, these scores are summed up for the highest scoring text block, which is considered as post author. However, the text block containing author name may contain other unnecessary texts like 'by', 'written by', 'and courtesy' etc. To filter these out, text-splitting methods, such as tokenization, n-grams and Inside–Outside–Beginning (IOB) tags from Natural Language Toolkit (NLTK) are used to pick human names.

Firstly, gather every chunk of text from the blog page filtering out the empty nodes and store them in a list. Then create a dataframe later on for storing records containing each block of text, its parent node and its score. For this purpose, initialize the following three lists and append values for texts and nodes for now. The scores are inserted after we have the calculations. Now, pass each block of text through the rules set by each pattern.

- The microformat rel="author" attribute in link tags (a) is commonly used to specify author of a post.

  <a href="author link" rel="author">Author </a>

- Author related keywords are used in attributes like class and id etc. in the node containing author name. It can also be present in the href attribute if wrapped between anchor tags (a) like:

  href = http://www.example.com/author/name

- To capture these patterns, we do the following.

*a)* Firstly, we create a list of all the keywords that may possibly refer to author, such as 'author', 'byline', 'source', 'writer', 'written', 'by', 'courtesy', 'contributor', 'originator', 'creator', 'builder', 'editor' etc.

*b)* We then create a string joining all attribute values of the parent node for each text node.

*c)* Finally, we look for the existence of each of the author keywords in any of the attributes we gathered in the string above.

- Sometimes author names are specified in between meta author tags (<author>…</author>). Therefore, we check whether the parent node for each text is 'author' or not.
- The parent html tag for the author text is most likely to be anchor tags (<a>…</a>). Therefore, this rule prioritizes the tag enclosed text blocks.
- The author information is supposed to be within a certain limit of text blocks. So, this rule emphasizes text blocks with a character limit of 50 or less in order to filter out larger chunks of text.
- In the end, all the scores multiplied by their respective weights are added up to yield the total score for each text block. The weights are calculated upon the precision and recall over the training data. We can use Bayes Classification algorithm or Random Forest model for this purpose. The more training data we have, the more it will lead to accuracy. For now, the weights are estimated based on a small set of data using Bayes algorithm. Finally, append score for each text block to the score's list we declared earlier.
- Create a pandas dataframe with lists (nodes, texts and scores) and select highest scoring text block as the post author data.

 ISBN: 978-1-61208-757-3 

- The extraction is not finished yet. The text block containing author name may contain some unnecessary text along with the author name like 'by', 'written by', 'and courtesy' etc. To remove the surplus and extract author name only, we use tokens, n-grams, IOB tags, parts of speech etc. from NLTK, which can recognize human names most likely of a person or organization.

*E. Extraction of post date*

Alongside the traditional rule based approach, two additional approaches are used to safeguard the extraction of the 'date' of an article. Earlier approach used to traverse through the html body and recognize the highest ranked text chunk based on certain rules. If this approach does not work, it will look for a date in the 'content' attribute value of potential meta tags in html head section. If failed to find the date here also, it will finally try to pull out the date from the post URL, which is a commonly used trend to describe the resource path of a blog post or article. Even after successful extraction of postdate, it may contain unnecessary texts like 'On', 'Published on' etc., or the date may appear in a variety of formats sometimes in human readable forms like 'Yesterday', '2 mins ago', 'Tuesday' etc. So, the extracted data are finally processed through a date parser to get rid of unwanted texts and store the value in any user-required format (currently in 'dd-Month-yyyy'). Methods are described elaborately below along with code segments. The process of how normalized text chunks are fetched and ranking methodology that chooses the based suited chunk will remain the same. This section focuses more on the postdate patterns. Going through the source code will give a better understanding of the sequence.

This approach follows the traditional method of ranking each block of text, based on five patterns.

- The postdate commonly appears to be within the time tags (<time></time>). If crawler wants to fetch full date time format, it can find it in the 'datetime' attribute of the time tag.
- Date related keywords may be present in attributes like class, id, title, and content etc. in the node containing postdate. To capture this pattern, first create a list of all the keywords that may possibly refer to postdate. Then simply create a string joining all attribute values of the parent node for each text node. Finally, look for the existence of each of the date keywords in any of the attributes that are gathered in the string above.
- Now a days, it is common for resource path of a blog post to show date format in the post URL. As a first step, extract the date portion of the URL and match it with every text chunk. The higher the match ratio, the more possibility of that text to contain 'date of the blogpost'.
- There is a chance that a single blog post may have multiple dates. For example, each comment may contain datetime, which crawler is not looking for. So we emphasize on the original date value by its depth in the html tree as postdate, it usually appears at the beginning, most of the time after the post title.
- The postdates are no longer in character length. Therefore, we can filter out larger chunks of texts by limiting the character length of the text to 50 or less than equal to 50.

From the above process, crawler only looks for the post date in the html body section. If this approach fails to extract, it looks for the date in attributes of meta tags defined in the HTML head section, which most often stores the publishing date of the article/blog post. The following steps gives an idea about what meta tags, a crawler should look for and where the postdate may be present.

Out of various meta tags, only look for selective ones. Postdate usually appears in the meta tags, which contain these four attributes:

{Name, property, itemprop, http-equiv}

- Then look for potential keywords for the date in the values of these attributes. For each of the above attributes, a separate list of keywords are defined.
- If a match occurs, we take out the date information from 'content' attribute value of the same meta tag.

On failure of capturing date in spite of applying the above approach, we look for the date information in the URL of the article, which is a widely used fashion to define the resource path of an article. For example,

http://www.example.com/2017/05/29/blog-post-title

To do this, isolate the resource path from the URL through URL parsing and use regular expressions to partition the date block from it.

Even after successful extraction of postdate, the chunk may contain unnecessary texts like 'On', and 'Published on' etc., or the date may appear in a variety of formats sometimes in human readable forms like 'Yesterday', '2 mins ago', 'Tuesday' etc. The dateparser module does an excellent job to process the extracted date, get rid of unwanted texts and store the value in common format (currently in 'dd Month, yyyy').

*F. Extraction of post content*

Unlike post-title, post-author or post-date, it is quite challenging to extract post-content to an accurate satisfactory level by rule-based pattern identification approach. Rule-based approach results in inclusion of boilerplates and larger-sized comments by users. However, there are few resources in the web like Dragnet, Goose, Readability, Eatiht, Boilerpipe etc., which does the same kind of job of extracting main content. In this case, we can

 ISBN: 978-1-61208-757-3 

use a tool "JusText" [15]. JusText is a useful tool to get rid of boilerplate content for example: navigation links, headers, footers and scripts from HTML pages. In a few cases, JusText cannot distinguish post-title, post-author and other unwanted texts and thus considers it as a part of the blog post/article-body. To avoid these, we can apply a few sanity checks and heuristics on the text returned by JusText. The procedure is explained step-by-step.

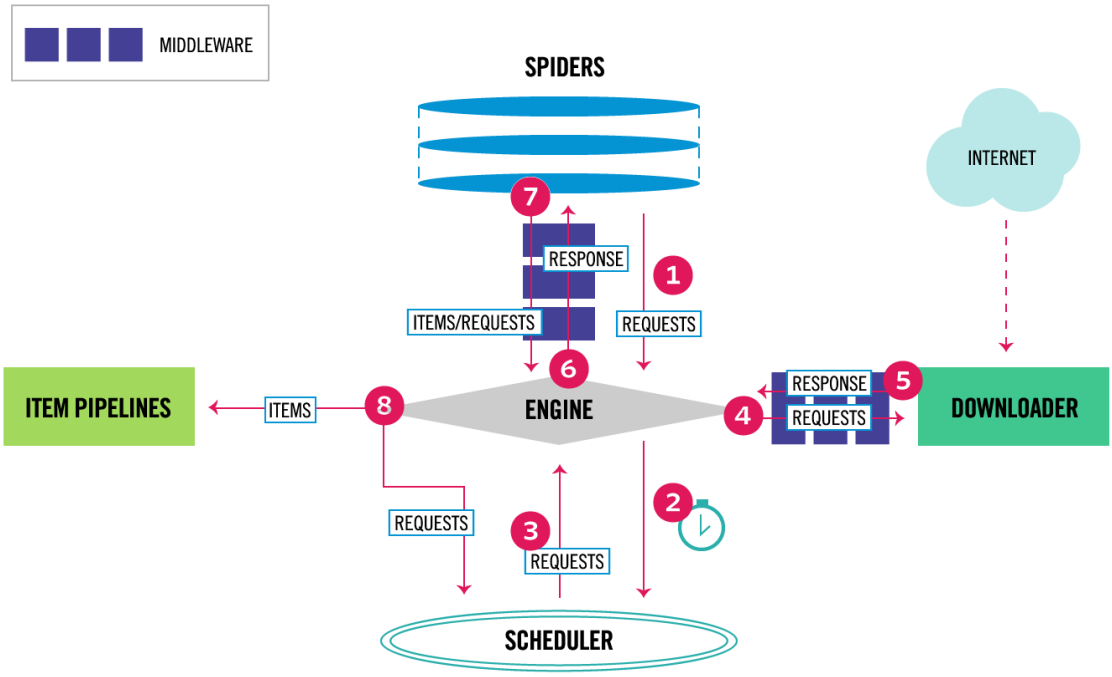


Figure 2: The architecture of crawling process by Scrapy [13]

- Justext can easily be installed via pip for either Python 2.6+ or 3.3+ [15].
- Then, requests can be sent to the blog-post URL to fetch pure texts from the response.
- Then comes the part of boilerplate detection. False cases in boilerplate detection are the texts we are targeting on, which further sanity-checks and heuristics should be performed.
- As mentioned earlier, these texts sometimes include other data like post-title, post-author or post-date along with the main blog-post content. So text-chunks can be checked to exclude these unnecessary data. By the time post-title, post-author and postdate are extracted, we can easily filter post-content out.
- Main blog-post contents are most likely to contain complete meaningful sentences. One tricky option to identify a complete sentence could be the text chunk ending with these punctuation marks: period (.), question mark (?) and exclamation point (!). This way unsolicited text chunks could be eliminated. We can also make sure the text chunks are trimmed on both right and left sides to ignore the dilemma of empty spaces.
- In very few cases, scripts are found embedded in the article body. These unnecessary scripts can be trimmed from the content.

The post content can also be extracted if the tag attributes of the html body for content contains the following property names in html body of the post.

['entry-content', 'article-body', 'article', 'articlebody', 'article-content', 'article-entry', 'post-body', 'post-entry', 'post-content', 'main-content', 'content', 'single-post', 'content-single', 'single-content', 'post', 'inner-content', 'page-content', 'postbody', 'material', 'material-body', 'article-main-content', 'material-body', 'materialbody', 'inner-post-entry', 'material-content', 'b-material content', 'article-inner-content']

### *G. Tools used*

The tools used for the implementation are as follows:

- Scrapy: An open source and collaborative framework built on Python for extracting the data from websites in a fast, simple way and yet extensible by design. It allows us to plug new functionality easily without having to touch the core [13]. The diagram above (Figure. 2 The architecture of crawling process by Scrapy) interprets the architecture of the crawling process.
- Python: Python programming language (V. 3.7.1)

 ISBN: 978-1-61208-757-3 

- MySQL Workbench: MySQL Workbench is a unified visual tool for database architects, developers, and DBAs.
- Elasticsearch: Elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine. It is developed in Java.

## IV. EXPERIMENT

For experiment, we matched the data we extracted through our generic crawler with the respective sample data crawled by Web Content Extraction (expected to be correct). We collected data from 10 following blogs with 26,579 posts.

- http://europeans101.blogspot.com/
- http://ukrainianlaw.blogspot.com/
- http://www.asianpolicy.press/
- http://www.rabble.ca/
- https://futuristrendcast.wordpress.com/
- https://informnapalm.org/
- https://uprootedpalestinians.wordpress.com/
- https://www.asia-pacificresearch.com/
- https://www.counterpunch.org/
- https://www.no-to-nato.org/

Here are the overall experiment results based on averages.
Precision: 0.9412558436393738
Recall: 0.9580803573131561
F-measure: 0.9489225566387176

## V. CONCLUSION

To sum up, in this paper we introduced the parsing of blog post pages of a blog for post attributes, using patterns that are automatically extracted from the blog's html patterns and implement a generic automated crawler for fast, robust and efficient blog data collection. We are still researching in what patterns, and to what extent blogs are interconnected. We also have great interest in analyzing the content of single weblogs. Due to this dynamic nature of blogs, we will face the long-term challenge of mining the blogosphere on a global scale. Even though the original implementation performed well along the milestones defined in the current crawler implementation, the accuracy of the crawler might not be 100%, but it can be smarter gradually adding more patterns to the data identification and by implementing machine-learning techniques.

## ACKNOWLEDGMENT

This research is funded in part by the U.S. National Science Foundation (IIS-1636933, ACI-1429160, and IIS-1110868), U.S. Office of Naval Research (N00014-10-1-0091, N00014-14-1-0489, N00014-15-P-1187, N00014-16-1-2016, N00014-16-1-2412, N00014-17-1-2605, N00014-17-1-2675, N00014-19-1-2336), U.S. Air Force Research Lab, U.S. Army Research Office (W911NF-16-1-0189), U.S. Defense Advanced Research Projects Agency (W31P4Q-17-C-0059), Arkansas Research Alliance, and the Jerry L. Maulden/Entergy Endowment at the University of Arkansas at Little Rock. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding organizations. The researchers gratefully acknowledge the support. We also thank Mainuddin Shaik and Muhammad Nihal Hussain for valuable suggestions.

## REFERENCES

[1] C. Fieseler, M. Fleck, and M. Meckel, "Corporate social responsibility in the blogosphere", Journal of business ethics, 2010 Feb 1, 91(4):599-614.

[2] Hosting Trubunal, "How many blogs are there" [Online], Available at: https://hostingtribunal.com/blog/how-many-blogs/, Last accessed on: Oct 6, 2019.

[3] M. N. Hussain, A. Obadimu, K. K. Bandeli, M. Nooman, S. Al-khateeb, and N. Agarwal, "A framework for blog data collection: challenges and opportunities", The IARIA international symposium on designing, validating and using datasets (DATASETS 2017), Jun. 2017.

[4] B. Mahar and C. K. Jha, "A Comparative Study on Web Crawling for searching Hidden Web", International Journal of Computer Science and Information Technologies 6.3 (2015), 1-5.K. Elissa, "Title of paper if known," unpublished.

[5] Newprosoft, "Web Content Extractor" [Online], Available at: http://www.newprosoft.com/web-content-extractor.htm, Last accessed on: Oct. 8, 2019.

[6] P. Berger, P. Hennig, J. Bross, and C. Meinel, 2011, "Mapping the Blogosphere--Towards a universal and scalable Blog-Crawler", 2011, IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing, IEEE, Oct. 2011.

[7] L. Wei-jiang, R. Hua-suo, H. Kun, and L. Jia, "A new algorithm of blog-oriented crawler", 2009 International Forum on Computer Science-Technology and Applications, Vol. 1, IEEE, Dec. 2009.

[8] BlogPulse, Wikipedia [online], Available at: https://en.wikipedia.org/wiki/BlogPulse. Last Accessed on: Oct. 8, 2019.

[9] N. Glance, M. Hurst, and T. Tomokiyo, "Blogpulse: Automated trend discovery for weblogs", WWW 2004 workshop on the weblogging ecosystem: Aggregation, analysis and dynamics, Vol. 2004, May 2004.

[10] L. Baker, "BlogPulse Search Engine" [Online], Available at: https://www.searchenginejournal.com/blogpulse-search-engine-launched-by-intelliseek/549/, May 2004, Last accessed on: 6 Oct, 2019.

[11] M. Lalmas, A. MacFarlane, S. Rüger, A. Tombros, T. Tsikrika, and A. Yavlinsky (Eds.), "Advances in Information Retrieval: 28th European Conference on IR Research", ECIR 2006, London, UK, April 10-12, 2006, Proceedings. Vol. 3936. Springer, Mar. 2006.

[12] P. Kolari, T. Finin, and A. Joshi, "SVMs for the blogosphere: Blog identification and splog detection", AAAI spring symposium on computational approaches to analysing weblogs, Mar. 2006.

[13] Scrapy [online], Available at: https://scrapy.org/, Last Accessed on: Oct. 6, 2019.

[14] Naive Bayes Classifiers [online], Available at: https://www.geeksforgeeks.org/naive-bayes-classifiers/, Last Accessed on: Oct. 6, 2019.

[15] "jusText 2.2.0 - Heuristic based boilerplate removal tool" [online], Available at: https://pypi.org/project/jusText/, Last Accessed on: Oct. 6, 2019.

 ISBN: 978-1-61208-757-3 